Box 7
Folder 11

ACM Conferences on the History of Computing, 1986-1987
Adele Goldberg papers
Conferences, workshops, and symposia
X5774.2010, *Box 7*

102739347

1 of 3

THE COMPUTER HISTORY MUSEUM
1 027 3934 7

ACM Conference on the

History of Personal Workstations



acm

Conference Proceedings

Papers Presented at the Conference

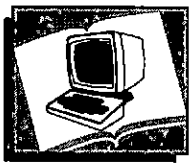
Palo Alto, California
January 9-10, 1986

Sponsored by the

Pacific Region of the
Association for Computing Machinery

ACM Conference on the

History of Personal Workstations



Conference Proceedings

Papers Presented at the Conference

Palo Alto, California
January 9-10, 1986

Sponsored by the

Pacific Region of the
Association for Computing Machinery

The Association for Computing Machinery, Inc.
11 West 42nd Street
New York, New York 10036

Copyright © 1986 by the Association for Computing Machinery, Inc., Copying without fee is permitted provided that the copies are not made or distributed for direct commercial advantage and credit to the source is give. Abstracting with credit is permitted. For other copying of articles that carry a code at the bottom of the first page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 27 Congress Street, Salem, MA 01970. For permission to republish write to: Director of Publications, Association for Computing Machinery. To copy otherwise or republish, requires a fee and/or specific permission.

ISBN 0-89791-176-8

Editor's Note

The papers contained in this proceedings are preprints distributed to conference attendees only. A final proceedings will be published by ACM as a book. Announcement of the availability of this book will be made in *Communications of the ACM*. For more information, contact: Publications Department, ACM, 11 West 42nd Street, New York, NY 10036.

John R. White
Xerox Palo Alto Research Center

Conference Committee

General Chairman and Program Chairman

Alan Perlis
Yale University

Conference Organization and Local Arrangements

John R. White
Xerox Palo Alto Research Center

Conference Support

Gloria Warner
Xerox Palo Alto Research Center

Proceedings

John R. White and Kathi Anderson
Xerox Palo Alto Research Center

Registration

Susan Mulhern
Xerox Palo Alto Research Center

Videotaping

Ken Beckman and Mark Chow
Xerox Palo Alto Research Center

ACM Headquarters Coordination

Helene S. Tannor
Association for Computing Machinery

Acknowledgements

A History conference is supposed to be a unique experience, and certainly from the viewpoint of conference organization it has been. A number of people have been involved in all phases of the Conference. The idea of bringing together researchers whose pioneering work dramatically influenced modern personal workstations originated with Bill Spencer, Manager of the Xerox Palo Alto Research Center. Alan Perlis agreed to contact potential speakers and convince them that a History of Personal Workstations Conference would be a unique opportunity -- something not to be missed. Not only did Al Perlis seem to know everyone in the field, so he knew who to invite, but he had been involved with the 1978 ACM SIGPLAN Conference on the History of Programming Languages, so he knew what perspective to convey in the invitation. The quality of the program is a credit to Al Perlis' persuasive personality.

Adele Goldberg suggested that ACM sponsor the Conference to give it an appropriate home and to help provide the infrastructure needed to pull off a meeting of this size. Throughout, Adele has been a constant source of good ideas. She suggested that the meeting be videotaped and agreed to edit the book that will represent the final conference proceedings. She has also played a leading role in using this Conference to launch a new series of ACM sponsored conferences on the History of Computing. Adele's enthusiasm and excitement have been infectious.

I want to thank a number of people at the Xerox Palo Alto Research Center for their help with this Conference. Kathi Anderson of the Computer Science Laboratory maintained constant contact with authors and helped immensely in producing the conference proceedings. Gloria Warner and Susie Mulhern provided excellent advice on local arrangements and took on full responsibility for conference registration. Videotaping a meeting of this size and editing a final set of tapes is a complex task that has been handled with cool, professional calm by Ken Beckman and Mark Chow.

Helene Tannor, SIG and Regional Conference Coordinator at ACM Headquarters, has been a tremendous help in all phases of this Conference, from negotiating an excellent hotel contract to providing advice on the conference proceedings. Not only was Helene a continual source of good ideas, but she was constantly able to point out potential pitfalls and brick walls that I had overlooked.

Finally, on behalf of everyone involved with the History of Personal Workstations Conference, I want to express our deep appreciation to the authors for taking time from current pursuits to rummage through ancient files and resurrect the facts, data and memories surrounding the early events that have led to modern personal workstations.

John R. White
Xerox Palo Alto Research Center

Foreword

Twenty five years ago computing was stationary, ponderous and centralized. Its dominant role was to serve the critical needs and purposes of organizations and the sciences. Today matters are very different. Computation is personal, ubiquitous and expansive. Power is being supplied at and to the fingertips of the individual. Computation is an amplifier of imagination and labor. The personal workstation is the gateway through which this power is supplied, harnessed, diversified, and distributed.

The transformation was not trivial, nor even inevitable. It was accomplished by people with energy, ambition and purpose, whose visions were supported by great technical expertise and insight. Their difficulties and triumphs are the stuff of which history is made. Thus it is important that we hear from them before the personal workstation becomes so packaged, so much a featureless part of our environment that its nature and value become invisible, a mere part of the overhead of daily living.

Alan Perlis
Yale University

John R. White
Xerox Palo Alto Research Center

ACM Conference on the History of Personal Workstations

January 9-10, 1986

Palo Alto, California

Thursday, January 9, 1986

Opening Session: 9:00 - 10:15

Welcome
William J. Spencer
Xerox Palo Alto Research Center

Keynote Address: Toward a History of (Personal) Workstations
Gordon Bell
Encore Computer Corp. 1

Session 2: 10:45 - 11:45

A Personal View of the Personal Work Station: Some Firsts in the Fifties
Douglas Ross
SofTech/MIT 19

Session 3: 1:00 - 3:00

Some Reflections on Early History (Abstract Only)
J. C. R. Licklider
MIT Laboratory for Computer Science 49

The Arpanet and Computer Networks
Larry Roberts
Net Express West 51

Session 4: 3:30 - 5:30

Mathematical Laboratories: A New Power for the Physical and Social Sciences
Glen J. Culler
Culler Scientific Systems Corp. 59

The Augmented Knowledge Workshop
Doug Englebart
MacDonnell Douglas 73

Banquet Presentation

The Dynabook - Past, Present, and Future
Alan Kay
Apple Computer, Inc. 85

Friday January 10, 1986

Session 5: 8:30 - 10:30

Personal Distributed Computing: The Alto and Ethernet Hardware
Chuck Thacker
Digital Equipment Corp. Systems Research Center 87

Personal Distributed Computing: The Alto and Ethernet Software
Butler Lampson
Digital Equipment Corp. Systems Research Center 101

Session 6: 11:00 - 12:00

The LINC Was Early and Small
Wesley Clark
Sutherland and Sproull 133

Session 7: 1:00 - 2:00

Hewlett-Packard and Personal Computing Systems (Abstract Only)
Chuck House
Hewlett-Packard 157

Session 8: 2:30 - 4:30

A History of the Promis Technology: An Effective Human Interface
Jan Schultz
Promis Information Systems 159

User Technology -- From Pointing to Pondering
Stuart Card and Thomas Moran
Xerox Palo Alto Research Center 183

TOWARD A HISTORY OF (PERSONAL) WORKSTATIONS (DRAFT)

C Gordon Bell
Page Farm Rd.
Lincoln, Massachusetts 01773

I originally accepted this keynote honor for five reasons: to respond to Alan Perlis' request (he told me I could present anything from a new taxonomy to personal reminiscences); second, to identify the important artifacts that should be preserved in The Computer Museum; third, to posit a framework of the history of workstations that can be written in the next century (we're all too close to create it); fourth, to summarize my own involvement on interactive computing including timesharing and large personal computers (both styles of use led to personal workstations); finally, to record my work on Digital's VAX Homogeneous Computing Environment that was totally announced last year. I led the architecture for this environment over the decade from 1973 to 1982 and was strongly influenced by the personal workstation computing movement including Ethernet. This work hasn't been described outside of product brochures.

INTRODUCTION TO THE CONFERENCE

In June 1976, an international conference was held resulting in a collection of views published in "The History of Computing in the Twentieth Century" (Metropolis, Howlett and Gian-Carlo Rota, 1980). The participants presented work up to the early 50's and thus had the benefit of 30 years of hindsight. The late mathematic's historian, Kenneth May, outlined the pitfalls of participant-written histories, commenting: "Historical description requires a time-lagged approach and means getting into understanding things as people understood them then, not as we undertstand them now." He also urged everyone to be open. Most autobiographies rewrite the past from the present perspective. Good diarists and notetakers have reference points. The best we can do is provide grist for the mill of the historian.

The History of Programming Languages Conference held in May 1977 (Wexelblat, 1981) provided a timeline to 1970 as context with detailed coverage of: Algol 60, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/1, SIMULA and SNOBOL. This covered a period of initial developments from 1954 (FORTRAN) to 1967 (APL) with evolutions up to the present. Ten to twenty years of hindsight gave them the ability to select topics.

If this conference were limited to a 15-year rule, applied by the Annals of Computing History, no one would be describing personal workstations. Only a few experiments existed, but people were building systems with multiple graphics terminals and a workstation industry on a profession-by-profession basis started to form. Twenty years ago, most of us were trying to make our newly designed interactive, timesharing system work reliably and economically. In my own case, having also worked on small machines that ultimately became the minicomputer, I saw two independent threads for economical, computing -- including interactive computing: large shared systems and dedicated small computers.

Personal workstations, like other man-made objects, appear strictly evolutionary going through the following stages:

50's	idea (documented article, proposal, movie, etc.) stimulated through early standalone use of small computers (e.g. LGP-30, G-15)
60's	breadboards to demonstrate the idea and selected use of large computers with graphic displays (e.g. DEC PDP-1, IBM 7090, LINC, TX-2)
70's	(early) limited use of interactive shared workstations using graphic display terminals connected to minicomputers; establishment of an industry to supply terminals and professional applications software
70's	(late) working prototypes of personal workstations with concept testing through use in a complete environment (Xerox PARC)
81	introduction of personal workstations by Apollo, SUN, Xerox, and first use by early adopters
83	full-scale use for selected professionals, many companies formed, JAWS--Just Another Workstation (term) coined
85	healthy industry with evolutionary product cycle and beginning shake out of suppliers
?	steady state supply to captive users
?	decline through replacement or superposition of functions in some other form of information processing appliance (e.g. conventional Personal Computers)
?	extinction

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0001 $00.75

hence, just as we should concentrate on tracing various workstations through these stages, we may also treat this conference about the first phases of development. More hindsight will be needed to write history

Waiting longer, which gives a future view, has the risk that extinction will come through agglomeration with conventional computing. Alan Perlis urged me to come to this conference because he feared this extinction. Conventional Personal Computers, that is the evolving IBM PC will become the de facto workstation for virtually all applications by 2005. The latest chip introduction by Intel strongly permits and supports this to occur, if history is any indicator.

The real workstation phenomena and industry is not well described or recorded at all because most of the products have come from industry where individuals are not rewarded for writing papers but producing and selling machines. For example Doug Drane and the ATEX system that produced law and typesetting offices doing essentially what a STAR does, only five earlier and less expensively; GE's genigraphic system for the graphic artist; Bill Poduska and the Apollo workstations; or Applicon, Calma, Computervision, DEC, Evans and Sutherland, IBM, REDAC, Wang, Datapoint's ARCnet, etc. No university (other than Utah and Brown) really participated until 1980. Noticeably absent are Dave Evans, whose direct contributions or indirect contributions through products at Evans and Sutherland, and graduate students who went onto create much of the science behind workstations. Ivan Sutherland's work was recorded. Xerox PARC was a noteable exception, which published its work in a delayed fashion and was marginally successful at taking the research into products in a commercially viable fashion.

We now need to focus and agree on some definitions, and dimensions making sure the various threads of personal workstation development are explored so the historical facts can be recorded properly, and eventually interpreted by real historians. We need to find the first use of the name, for example, the Second Edition of the Computer Encyclopedia, 1983, uses the word three times but in a somewhat different context. By the summer of 1983, the phrase JAWS (Just Another Workstation company) could be heard in the venture capital community when they realized they had funded too many companies to build personal workstations.

WHAT IS A (PERSONAL) WORKSTATION?

Terminals connected to large computers (supercomputers, mainframes and minicomputers) and utilizing interactive professional-applications programs are the clearest antecedents. These had, as direct descendants, terminals connected to shared, but dedicated, minicomputer systems in the mid 70's to deliver cost-effective computing. An overview of the classic machines and the classes they formed is given in Figure 1.

Figure 2 shows several paths leading to the personal workstation. Large machines (with large memories) provide the major impetus. Large shared systems allow the price per terminal to fall in

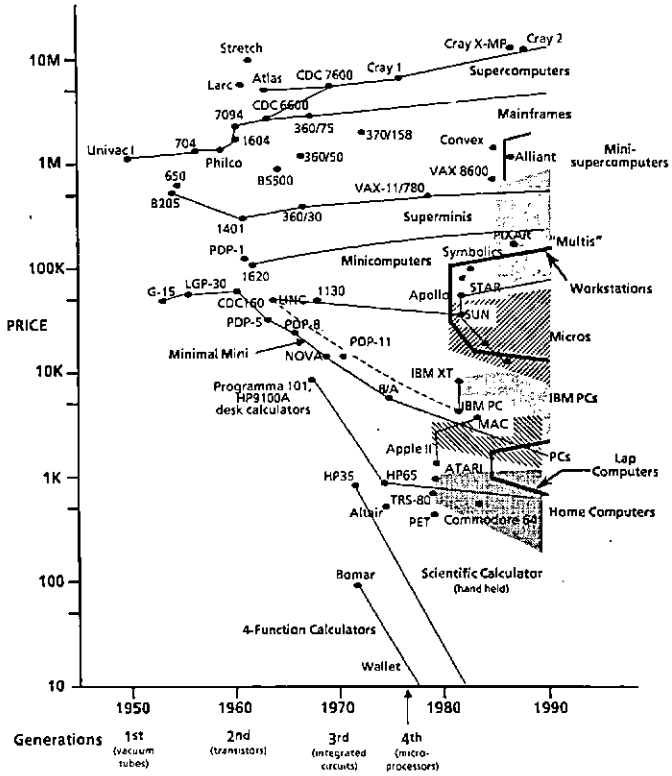


Figure 1. Computer system prices at introduction and classes versus time.

line with the salaries of the professional using the station. Four lines, corrected for inflation, depict the LINC price, the cost of a supported professional (e.g. engineer or scientist), the starting professional's salary, and a clerk's salary. The price of facilities such as a computer are "pegged" to these constants. The justification of equipment is usually based either on productivity or achieving some new level of

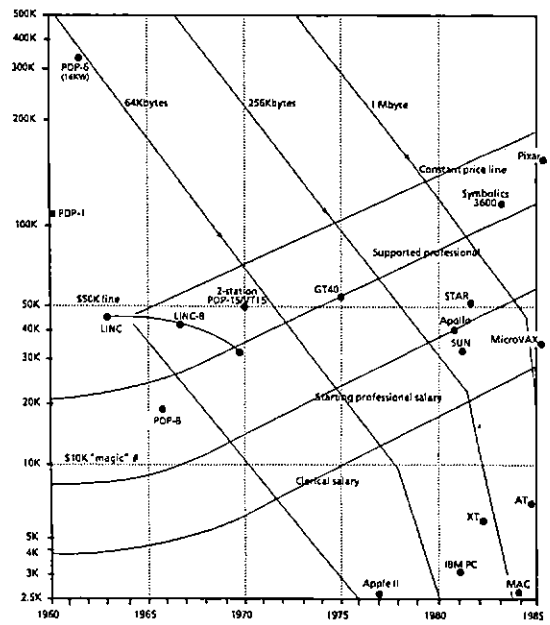


Figure 2. Workstation price versus time for various workstation introductions.

performance or capability. While salaries of clerks are greater than $10K, the workstation industry believes $10K to be magic, highly elastic, price barrier which will increase sales. In the later part of the 80's, large personal computers will easily have enough memory and capability to take over all functions provided by nearly any of today's personal computers or personal workstations. Constant price, very powerful, large workstations will continue to be built, but the mass market will be served by the rapid price decline inherent in the IBM PC evolution.

A personal workstation is a relatively large (>50 pounds) and expensive ($10,000 to over $100,000 in 1985) personal computer, with the appropriate transducers, used by a professional to carry out generic (e.g. calculation, mail and communication) and profession-related activities such as music composition, financial modeling or computer-aided design of integrated circuits.

Personal workstations are necessarily distributed with the person and interconnected to one another forming a single, shared (work and files) but distributed computing environment... the workstation environment. A workstation's location is either with an individual on a dedicated basis or in an area shared by several members of a group. This choice is dictated by the cost and size of the workstation, relative to the cost and value of the work.

Personal workstations appeared by name about 1981 with the concurrent appearance of:

- microprocessors with at least one-half megabyte of physical memory virtual memory addressing,
- memory chips of at least 64K bits providing primary memories of 1/4 to 1 megagbyte, permitting the use of large programs and construction of high resolution bit-mapped terminals
- disks of >10 megabytes
- local area networks for interconnecting the stations

Future descendants are likely to evolve from extended, lower-priced personal computers, interconnected via LANs. They would include appropriate transducers and corresponding professional applications programs.

A profession is any vocation, occupation or business, associated with work, including: secretarial, commercial, science, engineering, mathematics, and the arts.

A personal computer or pc is a self-contained computer with secondary file memory, and appropriate transducers to interface with people. A personal computer is used interactively by one person at a time, at a location convenient to the user and may "belong" either to the person or a group. A personal computer, for a given use, is self-contained, i.e. requiring no external program or data preparation units, permitting a user to go through various stages without external intervention.

The microprocessor, memory, and mass-storage technology appearing in 1975, lead directly to the personal computer industry. Early computers utilized the simple, single process, stand-alone operating systems developed for both interactive, timeshared computers and stand-alone minicomputers. Nevertheless the first personal computer, the LINC (Clark and Molnar, 1964) was built in 1962 long before its predicted technological time.

Personal computers can be used either in a

- personal or private environment encouraging separation of files, resources and work, with personal security whereby the only communication with other computers is via secondary memory (floppy) sharing, or transmission of messages via standard communications lines, or
- shared, workstation environment encouraging communication among the pc's, sharing of resources (eg. files, printers) and working on a single large work assignment or goal.

An IBM Personal Computer or PC is a particular personal computer utilizing the Intel 8086...80386 architectures and the evolving MS DOS operating systems. A PC may be extended and used in a workstation environment when interconnected via LANs and permitting access and sharing of facilities and work. Hence, constant-cost PC's are likely to evolve into the de facto personal workstations.

BELL MODEL OF MEMORY PRICE DECLINE, FORMING NEW COMPUTER CLASSES

Hardware technology improvements specifically in silicon and magnetic storage have been the sole enabling determinants of progress in computing... they are the "technological devils" that drive the formation of our industries. This is the economic basis that forms all computing classes, including all forms of workstations. Since personal workstations evolve from other computer classes, the entire hierarchy of computers must be understood. In effect, we get no product before its (technological) time!

In 1975, I observed that memory price was falling at a rate of 20% per year. Since it was a constant fraction of system prices, they also declined at the same rate. This 20% decline was based on the learning curves for manufacturing core memory manufacture and distribution. Thus, a computer would drop a factor of 10 in price, per decade.

In 1972, with the introduction of semiconductor memories by Intel, price began to decline a factor of four every three years, the cycle for new memory chip introductions. (The Four Phase company produced the first 1K MOS memory chips before 1970.) Table 1 shows the introduction of semiconductor memories and other devices, and the development of specific personal computers and personal workstation. The reduction in the price per bit of semiconductor memories amounts to a 36% price decline each year or a factor of 10 every 5 years, twice as fast as the original core memory (and discrete logic) based model.

In 1975, I also observed that memory size is the determinant of computer use and hence computer classes. That is computer power is proportional to memory size or memory size squared when a proportionally faster processor is used. Looking at the declining prices for various sized systems, one

Table 1 - Memory Chip and Microprocessor Introductions
With Resulting Personal Computer and Workstations

Year	Memory	Micro	Width	Memory Memory	Examples
1972	1K	4004	4		
1973		8008	8		
1974		8080/6800	8		
1975	4K	6502	8	4K-8K	Altair, IBM 5100
1976		Z-80	8		
1977					Apple II, TRS-80, PET
1978	16K	8086	16	16K-32K	
1979		68,000	16	16K-64K	Atari
1980		8088	8		
1981	64K			64K-256K	PC, Apollo, SUN 1
1982		186/286	16		Commodore 64
1983					LISA
1984	256K	68020	32	512K-1M	MAC, AT
1985		386	32		SUN 3
1986					?
1987	1M		32	4M	?

can see each computer class taking on the attributes and power of its higher order neighbor. Table 2 classifies what I then thought were the capabilities of various computers versus their memory sizes. I felt then, and still feel, that a one-megabyte address space for a process is the minimum size, and OK at least in terms of what we actually use. Anything smaller (e.g. 64Kbytes) is shear hell to use!

My goal in 1975 was to be able to use the past to project the evolution of simple computer classes to 1980 as the basis for the VAX evolution. Figure 3 shows the price of various systems versus their memory size. Note that a new computer class emerges when the price drops an order of magnitude. This evolutionary model was also described for DEC's hardware development (Bell, Mudge, and McNamara, 1978) shown in Figure 4. This evolutionary model turned out to be quite accurate, predicting both the Apple II (just 3 years away), low-cost shared micros, and the new class of mini-supercomputers that are beginning to emerge this year.

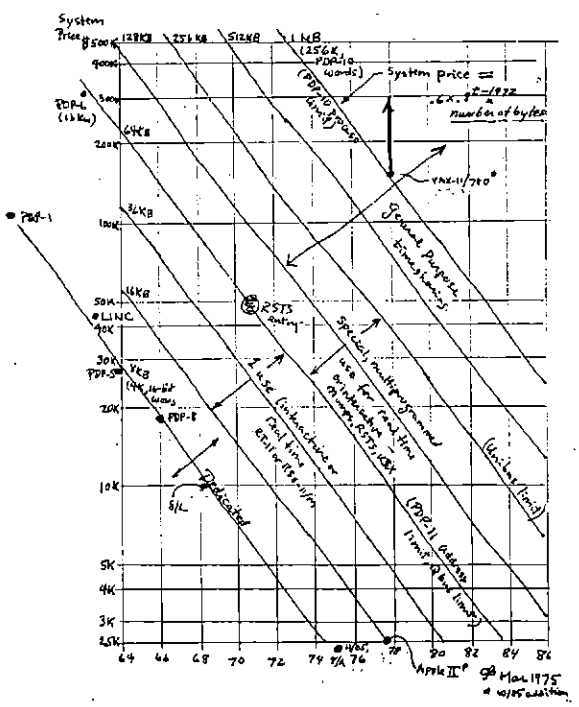


Figure 3. System price versus time for various memory sizes and system types (Bell 1975).

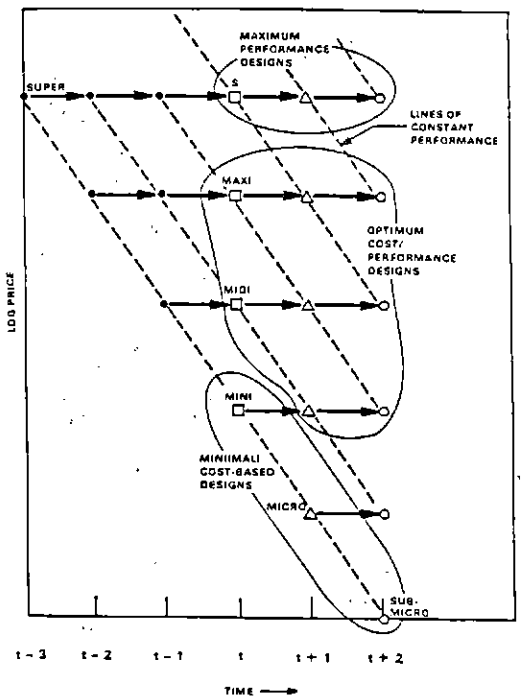


Figure 4. Price versus time for each machine class.

Since computer price determines the performance and therefore the economics and style of use, I began to observe at about the same time, that there were roughly three levels of computer use: central (mainframe), departmental (minicomputer) and personal. Dave Nelson, formerly at DEC, Prime and now at Apollo, extended this view to several other computer classes and introduced a model for price and weight (Table 3). These emerging smaller classes put significant pressure on interconnecting all the computers and forced the need for Local Area Networks (LANs). Figure 5 shows the style of use and environment for various computer classes. Note that personal workstations provide roughly the same capability of larger shared computers, and are large personal computers interconnected via LANs. IBM appears to have recently adopted this hierarchal view of computing and assigning meaningful work to all the levels, so it must be right (at least for a few years).

Table 2 - System Structure, Memory Size, and Resultant Use (G.Bell 1975)

Structure	Memory Range	Function (use)	
Dedicated (fixed-1 use)	4KB - 8KB	Interactive--e.g. POS cash register Real time--e.g. scope, traffic control, automobile	Special purpose, Fixed
Programmable (1 user)	16KB - 65KB	Interactive--RT11 (CP/M) Real time--RSX11S, M	Small scale, generality
Dedicated (multi-programmed) n-users)	65KB - 256KB	Interactive--MUMPS, RSTS, Trans.Proc. Real time--RSX-11M, D	Special purpose
Programmable (multi-programmed n-users)	128KB-1024KB	Interactive--IAS, TOPS 10, RSTS Real time--RSX-11D	Generality

Table 3 - Nelson+/Bell Computer Classes

Tier	Location	Names	Price($)	Weight(#'s)
0	wallet	calculator	10	.05
1	pocket	calculator, special function unit	100	.5
2	briefcase	kneetop, small personal	1K	5
3*	office	personal, large personal,"workstation"	10K	50
4*	department	shared micro, minicomputer, super-minicomputer, mini-supercomputer	100K	500
5*	center	mainframe	1M	5,000
6*	region	supercomputer	10M	50,000**

+ Founder and Vice President, Research, Apollo Computer Corporation
* Based on Bell's Distributed Processing Computing Model (circa 1975)
** Perhaps, if heat exchanger is included

NETWORKS AND THE SHARED AND PERSONAL WORKSTATIONS EVOLUTION

Although technology is a major variable that creates computer classes, the development of networks and changing patterns of use also influenced the evolution of personal workstations. The impact of the network and local area network on workstations is based on my perspective as chief architect of the ensemble of DEC products until 1983.

DECNET AND THE INFLUENCE OF ARPANET

The various DECNET architectures were initially created in 1974 by Stu Wecker while a member of DEC's research group. They were based on the notion of hierarchial computing according to use, concurrent with Digital's work on distributed processing. The imperative was to interconnect the hierarchy of DEC's minis and mainframes as tightly as possible forming a single computing environment.

The need for DECNET occured because Digital was proliferating timeshared and real-time minicomputers that had to have a way to communicate with one another and with its own and other central mainframes. DECNET was introduced in the following phases:

I	75	task-task, file transfer, file access; point-to-point; 8-bit net address; operating systems: RSX-11 D, M, IAS
II	77	improved system generation; used for first DEC engineering network (now several thousand machines at tens of sites; operating systems: rt-11, rsts
iii	80	routing, virtual terminal, full network management; operating systems: decstystem 20, vms
iv	83	large routing address, full remote rms files; c-terms; ethernet support
	85	portable DECNET (by Wecker's company)

The basic model for DECNET was ARPANET (Roberts, 1970), with the important difference that a network need not have specific IMPs or TIPs to do the packet switching for the worker computers. That is, the worker computers could do the packet switching for the network, and when the packet switching load became too high, one inserted packet-switching computers.

XEROX PARC ETHERNET/ALTO ENVIRONMENT, CARNEGIE-MELLON SPICE PROPOSAL

In 1979, Carnegie-Mellon published a proposal solliciting vendors for a personal workstation environment. Their statement, which had the effect of stimulating the design of various workstations was:

> "The era of time-sharing is ended. Time-sharing evolved as a way to provide users with the power of a large interactive computer system at a time when such systems were too expensive to dedicate to a single individual... Recent advances in hardware open up new possibilities... high resolution color graphics, 1 mip, 16 Kword microprogrammed memory, 1 megabyte primary memory, 100 megabyte secondary memory, special transducers,... We would expect that by the mid-1980's such systems could be priced around $10,000."

Although the concept was correct, the details of the machine were wrong because evolving technology was not considered. This led to selecting the wrong vendor, the late Three Rivers or PERQ Systems. Three Rivers' founder, Brian Rosen, a CMU grad who had worked at Xerox Palo Alto Research Center (PARC), designed PERQ as his version of the ALTO and D-series machines. Carnegie-Mellon may have taken on a leadership role in personal workstation environments by building its SPICE (Scientific Personal Integrated Computing Environment) on the PERQ and then contracting to build a similar environment for IBM, provided the SPICE derivative product is marketed.

The prototype for SPICE, and virtually all other distributed personal workstations environments (eg. MIT's Chaosnet and the personal LISP workstations that were the basis of LISP Machines Incorporated and Symbolics) was PARC's first 3-megabit-per-second Ethernet interconnecing Altos, operational as an environment in 1975 (Perry and Wallich, 1985; Pake, 1985). Datapoint's ARCnet with personal computers using the Intel 8080 was clearly the first commercial workstation environment. The Alto

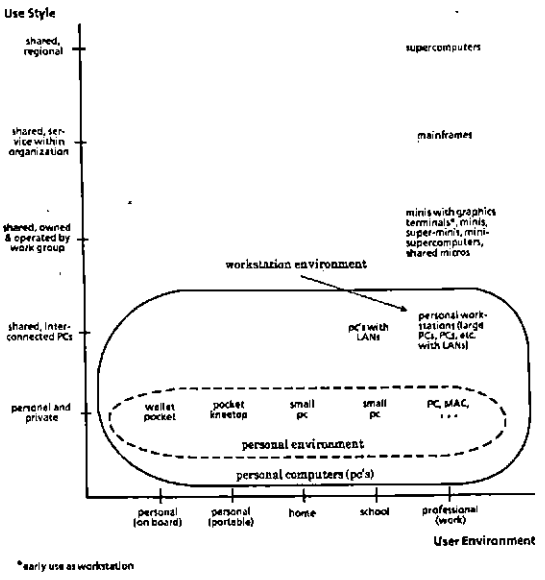


Figure 5. Computer use style versus user environment for various computer classes.

processor was a near derivative to the Data General Nova with an integrated bit-mapped display. It is important to note the PARC environment also included a timeshared computer based on the PDP-10 architecture, MAXC, which could be used for shared files and significant computation. In April 1981, XEROX introduced a product version of Alto, the STAR workstation and various file and print servers utilizing a 10 megabit per second Ethernet but without a real central computer. The PARC Ethernet was the forerunner of today's 10-megabit-per-second IEEE 802.3 standard (as originally drafted by Digital, Intel and Xerox).

THE DIGITAL VAX, HOMOGENEOUS COMPUTING ENVIRONMENT

The idea of a complete, Distributed Computing Environment, based primarily on the VAX architecture (but including the PDP-11 and DECsystem 10/20 computers) was presented to DEC's Board of Directors in December 1978 and approved as the strategy for guiding future product development.

The goals and constraints of this strategy is outlined in Appendix 1. The impetus was based on the early success of VAX, including plans to make a wide range of products to fill each of the 3 classes from MOS micros for personal computers to clusters of high-performance ECL VAXen. Furthermore, it was desirable to limit future evolution of the 10/20 and 11 lines, since both were providing essentially the same capability. The need for interconnection and the availability of LANs were critical as described above. Finally, computing was evolving from a centralized computing style to a more distributed computing environment as demonstrated by the Alto environment (but including high performance and departmental level computers).

The environment, shown in Figure 6 provided a wide range and hierarchy of compatible computers and ways of interconnecting them to provide users with generic (eg. word processing, mail) program

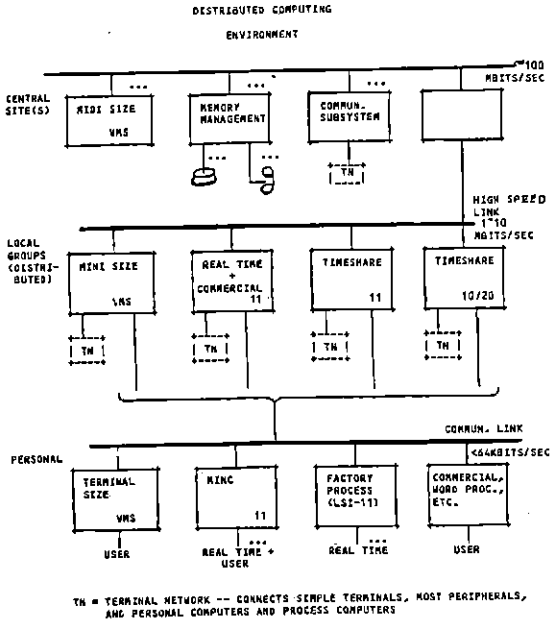


Figure 6. Digital distributed computing environment (Bell, 12-78).

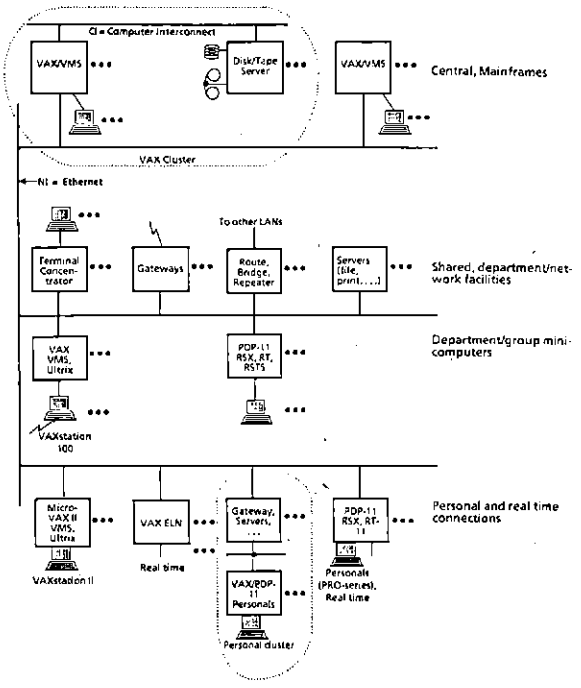


Figure 7. Digital homogeneous computing environment 1984.

development, and profession-specific computing using timeshared minicomputers or personal workstations. The goal was to provide the widest range of choices by having complete compatibility for where and how computing was to be performed without having to make a priori commitments either statically (purchase or installation) or even run-time to a particular computer system class (i.e. mainframe, minicomputer, team or personal workstation). By 1981, the original structure evolved to require the Local Area Network (Ethernet) for connecting all computers, initially called NI for Network Interconnect as part of the hierarchy of interconnects (CI=Computer Interconnect, NI=Network Interconnect, BI=Backplane Interconnect, and II=Interchip Interconnect; II did not materialize) as shown in Figure 7.

The design of the environment is substantially more than the design of a single range of compatible computers because diffent styles of user are required depending on the machine class and all the computers must be interconnected and work together in a multi-level hierarchy.

ETHERNET

While DEC had prototype LANs and proposals for LANs internally, I felt it was essential to have a standard. We hired Bob Metcalfe as a consultant and his missionary role was to make Ethernet the standard using Intel's chip design capacity and Xerox's patent and LAN experience. The IEEE got involved once they heard the idea of the LAN. They went on to facilitate the design of many other "standards", which of course, had the effect of diminishing the notion of a standard. Fortunately, the development of Ethernet crept into existence slightly more rapidly than the IBM Token Ring LAN which was introduced in October 1985, otherwise I doubt if there would be any significant use of LANs today. I understand about 30,000 Ethernets exist today.

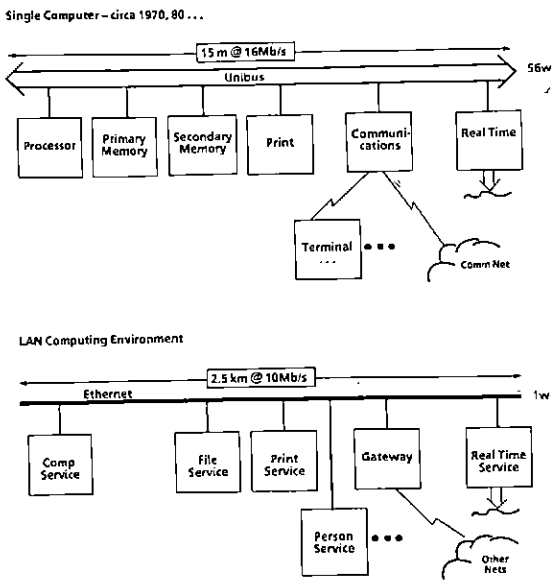


Figure 8. Unibus (c. 1970) and Ethernet (c. 1982) computer structures comparison.

In January, 1981 when Ethernet was being introduced by various vendors and being completed as an international stsndard, several of us from the developer companies (myself from Digital, Noyce from Intel, and Liddle from Xerox) made a broad appeal to the U.S. and European press and manufacturers on the importance of the standard. My theme was: "Ethernet is the Unibus of the Fifth Generation." We all argued that the standard was essential:

- to interconnect department level minicomputers and mainframe computers to one another, permitting file to file transfers, remote-terminal emulation, and distributed processing among the shared computers,
- as a switch for interconnecting terminals to the higher level computers within the environment,
- to centralize the gateway function in one place, versus requiring every computer to have protocol conversions to communicate outside the Ethernet environment,
- for building fully distributed computing environments, The PARC model of a computing system was to decompose computing into a series of functional servers such as printing, filing and with individual Workstations. The analogy with Unibus (circa 1970) as a computer versus Ethernet (circa 1981) as a computing environment is clear when comparing the two structures (see Figure 8).
- as a bus for building computer systems.

THE ENCORE CONTINUUM

With the formation of Encore Computer Corporation in 1983, our goal was to provide a complete environment, The Continuum (Fig. 9) like the VAX environment, but based on a single microprocessor. (Bell, 1985a). The "Continuum" provided both the distributed personal workstation style (described in the evolution, Appendix 1) and shared central computing using a multiple, microprocessor architecture, which I have named the Multi (Bell, 1985b). Ethernet provides the switch for interconnecting terminals to the Multimax using concentrators which pre-process terminal requests for communicating with other environments and for distributed-workstation environments. There are no terminal connections to the shared computers (something I have been trying to eliminate for years).

APOLLO DOMAIN

Dave Nelson, founder of Apollo and Domain architect (May 1980), was in the research group at DEC during the formation of DECnet and VAX. In the late 70's he went to Prime to work on the Prime token local area network and then left to develop the Apollo Domain architecture. Domain, introduced in March 1981, most typifies today's personal workstation because it provides a single addressing scheme for accessing all the memory (files) of a system. Dave credits me with stimulating this unifying architecture which is like the Unibus in terms of addressing.

WORKSTATION EVENTS, DIMENSIONS AND TIMELINE EVOLUTION

This section serves as a model for future historical research because it outlines, albeit incompletely, some critical events in workstation evolution.

The evolution of the physical workstation hardware is almost indistinguishable from the computer, and hence the evolution of personal workstations parallels and is directly coupled to the development of mainframes, minicomputers and microprocesssors. The principle difference is that workstations include a significant set of transducers to interface with humans and communication links, and additional user interface (software) advances.

PROCESSOR AND MEMORY COMPUTING ENVIRONMENT

The reinvention of the wheel and features within each of the computer classes has been nicely depicted by Burger, Cavin, Holton and Sumney (1984) as a timeline of the critical processor features (see Figure 10). The single most critical dimension of processing power is virtual memory size. The reinvention of the wheel of virtual memory size is shown in the Figure 11 from Siewiorek, Bell and Newell (1982) as each architecture reinvents and evolves the concepts of virtual memory.

VIDEO ENVIRONMENT

The evolution of displays has occurred along a set of dimensions which are relatively closely correlated with evolutionary time as follows:

- performance as measured in ability to display objects, including the ability to transform the objects
- hardware generation of the picture process. Myers and Sutherland's Wheel of Reincarnation shows the evolution of how pictures are controlled. (Figure 12 reproduced from Bell, Mudge and McNamara, 1978.) Note the evolution: direct control of the display object by the central processor; display list issued to a DMA controller or external controller; separate processor with loops, subroutines, etc.; arithmetic instructions added to aid dynamic displaying; use of virtual memory (beginning to occur to manage large pictures; and removing the display processor to form a separate computer.

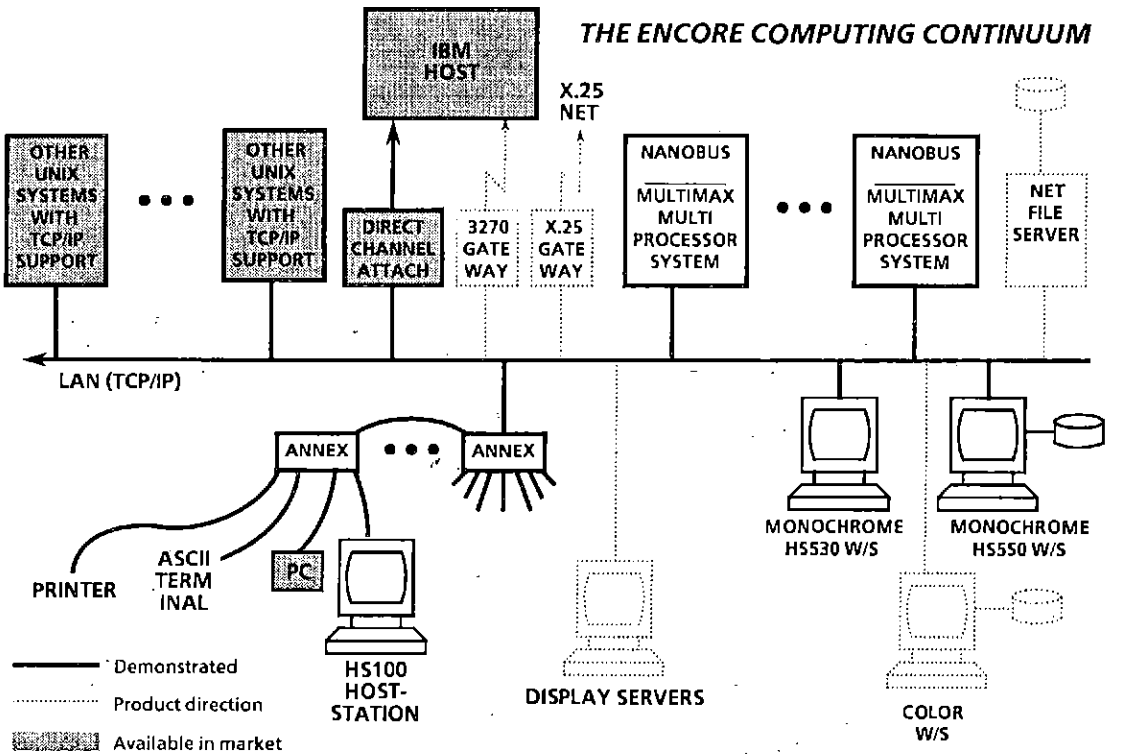


Figure 9. The Encore Computing Continuum.

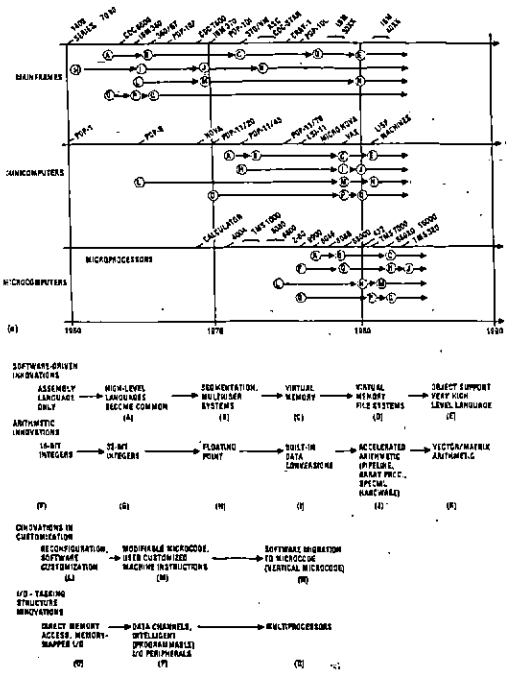


Figure 10. Computer architectural innovations for mainframes, minicomputers, and microcomputers from 1960 to 1990 (a) and the descriptive legend (b). (Taken with permission from Burger, Cavin, Holton, and Sumney, 1984.)

- resolution (in pixels or analog vector resolution)
- number of bits per pixel including use for monochrome/color
- display data-types
  - points, characters, lines, curves,
  - 3D line drawing
  - wireframe meshes, 2D surfaces, 3D surfaces,
  - shading, sculptured surfaces, constructive geometry
- translation of data-types
  - fixed
  - translation, scaling, rotation,
  - 2D clipping, 3D perspective, 3D clipping
- standard interface evolution: direct control, Siggraph CORE Library) VDI, CGI, NAPLPS, etc.
- windowing of output images
- character, line, picture and 3D image input

TACTILE AND MANUAL POSITION OR LINE INPUT

This category includes: keyboard, 2D and 3D light pens, direct touch screen, tablet, digitizing tablet, knob, joystick, track ball, mouse, etc.

PAPER OUTPUT

This section must include the evolution of: conventional printing and plotting along with early film recorders to capture images and extending to modern dot matrix printers of all kinds for monochrome and color.

VOICE I/O

While today's personal workstations do not permit substantial voice communication, voice input such as the Kurzweil voicewriter promises to add a significant dimension for human communication. The critical voice "firsts" for use with personal workstations are relatively sparse at this time.

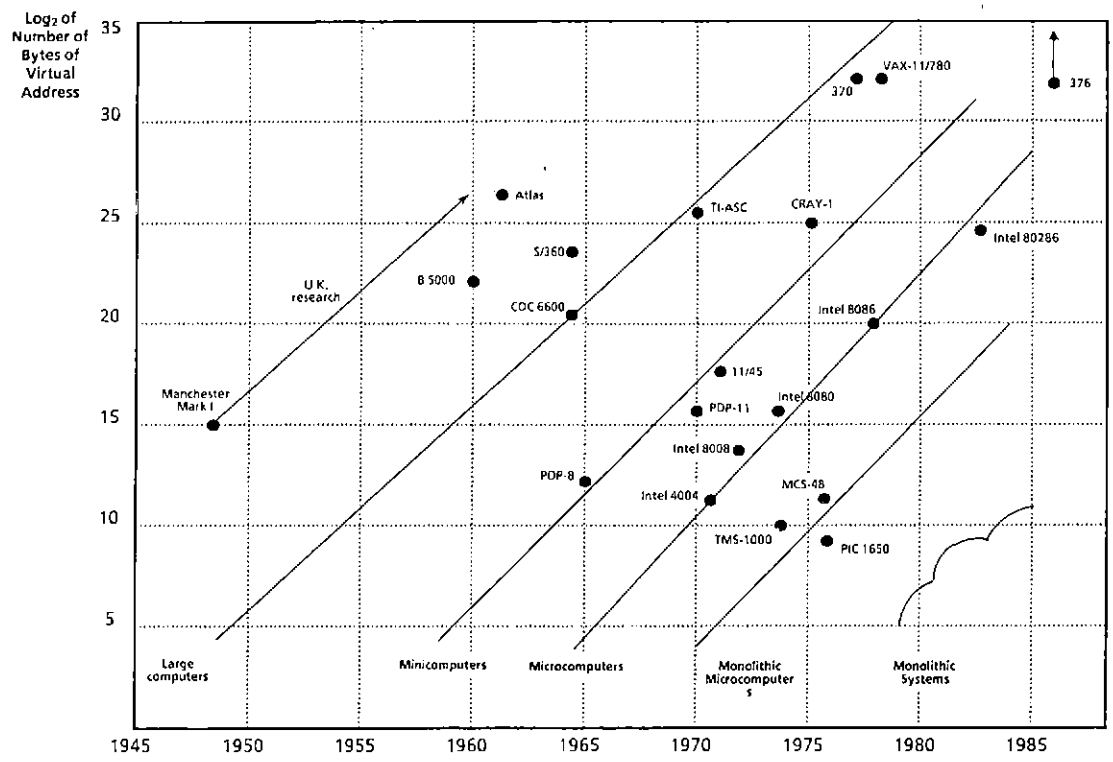


Figure 11. Virtual address space evolution for various computers and computer classes.

LINKS, SWITCHES AND NETWORKS
Links, switches and communications networks have influenced the formation of computer networks and computing environments. The previous section described their influence on the formation of the DEC computing environment. Others should be traced.

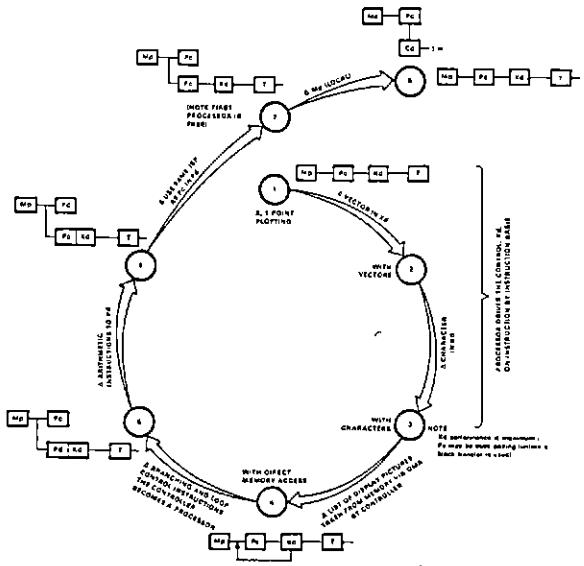


Figure 12. Myer and Sutherland wheel of reincarnation for display processor evolution.

SPECIAL PURPOSE I/O AND ROBOTS
The ability to handle arbitrary signals from other environments is crucial to the formation of various profession-specific environments (e.g. music, scientific experiments).

OPERATING SYSTEM, DATA-BASE, LANGUAGE, AND GENERIC ENVIRONMENTS
Traditional computer science history has laid claim on operating systems, databases, and languages including their application to personal workstations. For example, the History of Programming Languagea conference constructed an excellent timeline of the various languages including those for specialized environments such as interactive computing and specific application domains such as statistics, science and engineering.

Generic environments which users of traditional, timesharing computers have required include:

- TIMESHARING ORIENTED WITH TEXT AND GRAPHICS TERMINALS (SLOW LINKS)
  - text editing, mail and network mail
  - drawing and graph plotting
  - shared word processing
  - database access
  - transaction processing
- SHARED WORKSTATION AND PERSONAL WORKSTATION ORIENTED (FAST LINKS)
  - multiprocess control including control, windows
  - spread sheets
  - drawing and painting

PROFESSION SPECIFIC WORKSTATIONS

In addition to generic applications programs (eg. word processing and mail), each profession requires an environment to support and "understand" the work of the professional. This may take the form of special hardware (eg. high-resolution mega-color scopes for the graphic artist, high precision analog output for the musician, or fast simulation processors for the VLSI designer) in addition to specific programs that understand and have knowledge or expertise of the work being carried out. The following taxonomy provides an outline of the environments that have developed quite independently and that historians should research:

- SCIENTIFIC, MEDICAL AND MATHEMATICS WORKSTATIONS
  - life science with transducers appropriate to the measurement task
  - medical (eg. radiologist, cardiologist, bone surgeon)
  - geologist, petroleum engineer
  - physics experiment control and analysis
  - statistician and mathematician
  - signal analysis, encryption/decryption
- ENGINEERING WORKSTATIONS
  - electrical or electronic schematic (2D line drawings)
  - VLSI physical design 2-1/2 D
  - mechanical drafting
  - design of sheet metals or piping (2-1/2 D)
  - architectural and structural engineering
  - general mechanical design of arbitrary surfaces for analysis and fabrication
- SOFTWARE AND KNOWLEDGE ENGINEERING WORKSTATIONS
- ARTS (TYPESETTING, GRAPHICS ARTS, MUSIC, ETC.) WORKSTATIONS
  - publisher of: newspaper, magazine, book
  - scholars including: historians, linguists, literary scholars
  - graphic artist, illustrator, cartoonist, choreographer
  - composer
  - artistic film production
- COMMERCIAL WORKSTATIONS
  - general clerical and transaction processing, office word processing, executive
  - financial personal workstations for: banking, insurance broker
- SPECIAL CONTROL-ORIENTED WORKSTATIONS
  - air traffic control
  - process and plant control
- TRAINING WORKSTATIONS - (and are these workstations?)
  - simulation of industrial plants, aircraft, etc
  - Computer Aided Instruction such as Plato

SOME DISPLAYS, PERSONAL COMPUTERS AND WORKSTATIONS I HAVE KNOWN

This section describes various displays connected to computers, which were used as either personal computers or workstations. The MIT and Lincoln Laboratory displays embedded in Whirlwind, TX-0, TX-2, and LINC were among the earliest personal computers. Since these computers (except LINC) were one of a kind, their impact was as prototypes for subsequent developments. One direct impact was in the formation of the DEC computers, many of which were used as personal computers.

The first DEC display, Type 30 for the PDP-1 introduced in 1961, had instructions that permitted direct plotting of points and optionally, characters and vectors. As DEC evolved and built timesharing computers with operating systems which protected the user from accessing input/output equipment directly, the ability to have highly interactive terminals and workstations decreased.

Nearly all of the DEC computers were used as components for building interactive, "workstation-style" systems. DEC and Original Equipment Suppliers (OEMs) built and sold "workstation-style" systems for use in the sciences, printed circuit layout, mechanical and architectural drawing, typesetting and office automation. These pioneering applications (circa 1970) operating on dedicated timesharing systems served as the basis for much of today's evolving workstations.

MIT WHIRLWIND, LINCOLN LABORATORY TX-0, TX-2, AND IBM AN/FSQ7 (SAGE)

During Whirlwind's first year of operation, 1950, Bob Everett published a paper that showed a picture of its display (with an attached camera) and listed some actual problems carried out on the machine:

1. An industrial production problem for the Harvard Economic's School
2. Magnetic flux study for Whirlwind's magnetic storage work
3. Oil reservoir depletion studies
4. Ultra-high frequency television channel allocation for Dumont
5. Optical constants of thin metal films
6. Computation of autocorrelation coefficients
7. Tape generation for a digitally-controlled milling machine

Whirlwind was initially established as a large specialized simulator to the Airplane Stability and Control Analyzer (ASCA). The actual use turned out to be a prototype for the SAGE (Semi Automatic Ground Environment) system for air defense and SAGE, in turn, was the basis for air traffice control. The first successful use of Whirlwind for air defense occurred on April 20, 1951.

MIT's Lincoln Laboratory, established in 1951, moved into its quarters in Bedford, Massachusetts, in the summer of 1953. Lincoln built the TX-0 computer to test the use of transistor circuitry and a large core memory and then the TX-2 for large-scale computing experiments. Both had a point-plotting display with 10" x 10" area and 1K point resolution, light pen, camera, switch input, and abilities to interconnect arbitary i/o devices. Hardware innovations of the TX-2 include: addressable magnetic tape for a filing system; The Lincoln Writer, a typewriter for engineering/scientific use; and the TX-2 multiple sequence operation for rapid context switching. While the TX-2 was initially a personal computer, it operated under control of a timesharing operating system by the mid 60's. The TX computer circuitry was virtually identical to the logic and laboratory modules that DEC sold in its first four

years, prior to the introduction of the PDP-1 in 1961.

These machines used as personal computers pioneered numerous applications, the most famous being Sutherland's Sketchpad.

I spent a brief but wonderful period (1/59 - 6/60) using TX-0 for speech research and writing the program to do "Analysis by Synthesis", (Bell, et al 1961) which took as input speech spectra directly from a filter bank connected to TX-0. I also spent six months designing an IBM-compatible tape TX-0 and exploring hybrid computation (an analog computer used with TX-0). Peter Deutsch, then age 12, acted as a relatively patient user consultant and helped me debug my macros. Gwen Bell entered data on land use in the Boston area, and I wrote a program to display and explore this data. The program was demonstrated to the city-planning faculties at MIT and Harvard. Ten years later, Harvard was able to turn the clock back over 30 years when they established their computer graphics laboratory to do a simpler version of computer mapping, using punched cards and line printers.

DEC PDP-1 DISPLAYS: TYPE 30, 31, AND THE FIRST COLOR DISPLAY

The PDP-1 continued in the tradition of the "MIT personal computers". The PDP-1 had three displays. The Type 30 point plotting display, which took 50 microseconds to display a point on the 10" x 10" tube with 1K resolution. This was one of the first displays connected to a commercial computer, converting it into a personal computer for various scientific data analysis applications. Type 30 was extended to include a character generator which was controlled by loading a 36 bit, 5 x 7 bit raster specifying the character. The circuitry, designed by Ben Gurley, formed the basis of the 3XX displays. The DEC Special Systems Group designed an elaborate display, PEPR, for bubble chamber photograph analysis for MIT using the basic display circuits.

Type 31 designed by Gurley was a 5" high precision display with 4K point resolution. Only two? were built, the first being delivered to Lawrence Livermore Laboratory (LLL, now LLNL--the "N" for National) for use on their PDP-1. The Livermore PDP-1 had every option that DEC had proposed to build including Remington Rand tapes, and was used as a front end and exloratory PC for the large computers including Stretch snd LARC. Ben Gurley left DEC and with Ed Fredkin established Information International Inc.(III). The first III product included a high-resolution display for various image input/ouput applications. The basic high resolution display is still in existence.

Two? PDP-1 Color Displays were built, one of which was delivered to the Air Force Cambridge Research Laboratory (the military counterpart to Lincoln Laboratory). The color display was built by modifying a standard RCA color television for point plotting.

At the Fall Joint Computer Conference in 1961, Ed Jacks, from General Motors, spent almost the entire conference watching people use the PDP-1/Type 30 draw and doodle using a program I had written. The display influenced Ed to tackle Computer Aided Design, later using the IBM 7090 with a connected display. The classic program SPACEWAR! was written on the PDP-1 in 1962, by Steve Russell. Today SPACEWAR! continues to be the center to settle litigations about the design of computer games.

LABORATORY PC'S: LINCOLN LABORATORY LINC, DEC PDP-12, AND MINC

Wes Clark and Charles Molnar continued the MIT tradition of building personal computers, and LINC was designed for wider scale use for the life sciences, by being lower cost (about $40K in 1962). The LINC was designed using mostly DEC modules. The original LINC had two 5" displays, a keyboard, 1 or 2Kw, 12-bit memory and two LINCtapes. Each unit had approximately 256 Kbytes of addressable tape memory. I believe LINC was the first production personal computer for scientific use (about 50 were produced, 21 by DEC). Its significance was its completeness at the low cost, yet being open-ended.

DEC went on to build 140 LINC-8's beginning in 1967 (designed by Wes Clark and Dick Clayton) and around 1,000 PDP-12's beginning in 1970 (Bell, Mudge, and McNamara, 1978).

DEC 338, 339, AND 340 DISPLAYS

The 338/PDP-8 display/personal computer, introduced in 1967, was used as a front-end computing terminal. The 338 and 339 (for the 18-bit computers) was, I believe, the first display processor (Bell, Mudge, and McNamara, 1978). It took a display program in the PDP-8's primary memory, which specified points, character strings, vectors, subroutines and other program-control instructions, and permitted completely autonomous operation of the display, independent of the central processor. The 339 was used with the 18-bit computers, especially PDP-9. Both systems were used for standalone data analysis and for display terminal front ends to large computers.

A special version of the 340 was designed on a Saturday morning with Ivan Sutherland for his use within the government just prior to joining ARPA. This PDP-7/340 had every conceivable keyboard and input device we could think of, including switches, trackballs, joysticks, lighted buttons, etc.

DEC PDP-15/VT15, AND GT4X "WORKSTATIONS"

The PDP-15/VT15, introduced in 1970, costing about $90K was used as a standalone for one or two users (to reduce the cost to $50K per terminal using a $15K graphics terminal). About 30% of the systems had two terminals. It took about 100 microseconds to draw a vector using the DDA technique. The PDP-15 had from 8 Kw to 32 Kw of memory. Computer Aided Design system for Printed Circuit layout written snd marketed by REDAC (U.K.) was a key application, although it was used extensively for scientific-research applications.

In January 1973, the GT40 was introduced at a price of $25K with 16 Kw of memory based on the lower cost PDP-11/05. The well-known program, Lunar Lander, was written by Jack Burness for the product introduction. The relatively slow speed of the display, coupled with a limited memory and configuration of the 11/05 limited the application to a pre-processor for time-shared systems doing CAD, etc.

The GT44, was introduced in June 1973 at a price of $38K, using the same display processor as the GT40, but with an 11/34 computer and RK05, 5 Megabyte disk. The GT44 was used as a complete, standalone computer.

The GT48, introduced in October 1975, selling for $55K, was a high performance personal computer, using the 11/34, with memory up to 128 Kw. Vectors were generated using analog circuitry at a rate of 100,000 vectors per second.

THE PDP-8 WORD PROCESSING PERSONAL COMPUTERS
In addition to the early use of the PDP-8, with the 338 and various storage tube displays, the PDP-8 began to be used extensively 10 years after its birth for the DECmate series of word processing terminals. The PDP-8 was based on the PDP-5 (1963) 12-bit architecture, and influenced by the CDC 160 and LINC, addressed up to 32 Kw (later expanded to 128 Kw. The following word processing personal computers used the PDP-8:

Year	Model	Price	Comments
1975	DS310	$16K	8/A, desk, 8" floppy and VT52 terminal,
1976	WS200	$40K	8/A, cabinet, 1-8 users, 5 Mbyte disks,
1977	VT78	$15K	Intersil 6100 PDP-8 chip, embedded in VT52, 8" floppies
1981	DECmate I	$12K	6120 chip, embedded in VT100, 8" floppies
1983	DECmate II	$3.5	integrated unit, 5" floppies, bit mapped,
1984	DECmate III	$2.4	Z80/8086 and 5" winchester disk options

By 1984, the 21st birthday of the basic architecture, over 100,000 PDP-8 connected terminals or personal computers were used for word processing. More PDP-8s were built in 1984 than any other year. In a very large fraction of the use, the personal computers connected with larger departmental-level VAX computers for electronic mail, filing and auxiliary word processing for a true, shared workstation environment.

LSI-11 BASED PERSONAL COMPUTERS: TERAK, PDT 11/150, AND VT103
The LSI-11 (11/03), introduced in 1976, with a smaller printed circuit board form factor (approximately 10" square) and using the Qbus proved to be an important building block for Digital's subsequent personal computers.

Terak was founded in 1975 to exploit the notion of a bit-mapped graphics computer and used the LSI-11 (11/03) and subsequently the 11/23. The 8510, first delivered in 1976 to Ken Bowles at the University of California, San Diego for the development of UCSD Pascal, was developed indpendent of the Xerox Alto. The system operated with up to 28 Kw of memory, used 256 Kbyte floppies, and the bit-map display's resolution was 320 x 240. About 3000 were built and sold at a price of $5000 to the education market.

DEC introduced the PDT 150 as a small computer with two 8" floppies and 32 Kw at a price of less than $10K. While all the DEC marketing groups debated the price and their role in distribution, the personal computer industry formed. DEC was able to discourage Dan Bricklin from using PDT to build Visicalc. (He went on to use the newly introduced Apple II.) The VT103, introduced in 1981, was based on the VT100, which had been introduced to accept the Qbus, LSI-11/23 modules.

DEC TYPESETTING TERMINALS: VT20 AND VT71
The PDP-11, introduced in 1974 and 1977, was used as a base for the VT20 and VT71 typesetting terminals. These had a programmable character set of 256 characters, used 15" portrait oriented tubes, and communicated with the host via 9600 baud asynchronous communication. The VT20 was based on the 11/05 and could handle two CRTs, while the VT71 used the 11/03 and sold for $8 K. These terminals connected to host PDP-11's which carried out the main typesetting editing functions.

DEC VAXSTATIONS: 100, I, II AND 500 SERIES
The introduction of high resolution, bit mapped terminals, connected to VAX occurred quite late in VAX's life. A number of attempts were made, including SUVAX (for single user VAX) providing both monochrome and color, operating in 1980.

The VAXstation 100, introduced in May 1983, was a high resolution 19" terminal with bit-mapped graphics, which connected to any of the VAX computers by a high speed fiber optic link connected to the VAX's I/O bus (the Unibus).

VAXSTATION I and II, based on the MicroVAX I and II, were introduced in October 84 and June 1985, respectively. Both units sold for approximately $20K, with 19" bit-mapped CRT, using the Qbus for interconnecting I/O equipment.

The VAXstation 500 family was introduced in October 1985 with 1280 x 1024 pixel resolution color permiting 2D and 3D wire frame and 3D shaded solid images.

CONCLUSIONS
Personal workstations have evolved from many directions and along many dimensions. This conference barely scratches the surfaces of a few dimensions. We should endeavor to be as complete and accurate as possible in what we provide and in going from these papers to a more widely available publiction.

The conference has served to convince me that the critical dimensions of personal workstation evolution should be explored and documented by historians.

ACKNOWLEDGEMENTS
Dave Fanger, Encore Computer, did the figures on a Xerox Star. Designers of DEC products supplied product information: John Clarke and John Kirk (PDP-8), Herve Lavoie (VT15 and GT4x), Jim Milton (typesetting), and Stu Weckar (DECnet). Dave Nelson, Apollo, contributed a table and much to my understanding of workstations. Gwen Bell helped clarify my thoughts well enough to get a first draft, which she then edited. Mary Jane Forbes prepared the final manuscript.

APPENDIX I. THE DIGITAL VAX, HOMOGENEOUS COMPUTING ENVIRONMENT

TECHNOLOGICAL AND MARKET BACKGROUND

BATCH MAINFRAMES FOR CENTRAL SERVICES

In the first two computer generations, 1950-1970, computers were used in batch processing under the name of mainframe computing. During the 70's the mainframe began to be used almost interactively from remote job entry terminals as "glass key punches". The general direction is to have larger mainframes and larger terminal networks that interconnect to a single computer by an array of front-end computers. When more power is required, more switching computers are connected to several mainframes each of which perform a particular function. Attached dual processors are used to provide increased power for what is fundamentally a single system. Over time the evolution will be to small-scale multiprocessors for incremental performance and higher availability.

MINICOMPUTERS AND TIMESHARING FOR A GROUP

In the mid-60's both minicomputers and timesharing were developed at Digital around the PDP-8 and PDP-10 computers, respectively to lower the cost of computing. Minicomputers were initially used as components of real-time systems and for personal computing. The LINC minicomputer, developed at M.I.T.'s Lincoln Laboratory, was the first personal computer, providing a personal filing system and the ability to write and run programs completely on line.

Timesharing started out as a centralized mainframe facility for a large group with the early demonstration at MIT with CTSS, followed by the introduction of products by DEC (PDP-6), GE (using Dartmouth's BASIC) and Scientific Data Systems 940 (based on the Berkeley Timesharing System). Access was via individual Teletypes which were eventually replaced by cathode ray tube terminals, or "glass Teletypes". By the mid-70's low cost PDP-11 timeshared computers began to be used by separate groups and departments to provide "personal computing". By the mid 70's a number of minis used high performance graphics terminals for CAD/CAM, typesetting and other applications requiring graphical output or fast response. These minis were the forerunner of today's distributed, personal workstation environment.

In the early 80's, low cost disks and large memories permitted two evolved computer structures: the 32-bit supermini, and the microprocessor based "team computer". The supermini had all the power of its mainframe ancestors, especially the critical 32 bits to access memory. The "team computer", based on modern, powerful microprocessors, is simply much lower priced (eg. $15,000) providing "personal computing" at a price below personal computers.

WORKSTATIONS

The fourth generation appeared in 1972 with the microprocessor. With the second 8-bit generation microprocessor, floppy disks and 16 Kilobit semiconductor memories (circa 1976), Personal Computers were practical and began to be manufactured by Apple, Commodore, Radio Shack, etc. With 16-bit microprocesor (measured by data-path) and 64 Kbit rams, the second generation of PCs appeared in the early 80's which were suitable for building personal workstation environments.

Table 4 summarizes the characteristics of these styles of computing.

In 1979, Carnegie-Mellon University wrote a proposal for personal computer research to implement a workstation environment, and declared: "The era of timesharing was ending." Today's powerful personal workstations, such as the Apollo or SUN workstation providing a large virtual address, connected with shared facilities on a local area network characterize this type of machine and computing environment.

PERSONAL COMPUTERS CLUSTERS (I.E. THE WORKSTATION ENVIRONMENT) AS AN ALTERNATIVE TO SHARED COMPUTERS

In the mid 70's, Xerox Research Park researchers developed and provided itself with a personal computing environment consisting of powerful personal computers all linked together via the first Ethernet cable (3 Mbits), and created the notion of the Local Area Network. Their network had various specialized function servers, including a shared central computer that was compatible with the DECsystem 10, for archival memory and large scale computation.

Figure 13a shows the hardware and software of a multiprogrammed computer used for timesharing, and the corresponding structure of a Personal Computer Cluster consisting of functional services and interconnected by a common interconnect which provides basically the same capability. The timeshared system has a central memory containing various jobs connected to terminals and operating system which attends to the users and handles the particular functions (eg. real time, files, printing, communication). Personal Computers are connected to timesharing systems aa terminals. By comparing the shared system with the systems formed

Table 4 - Computing Style Characteristics (Circa 1978, G.Bell)

STYLE*	CENTRAL	DEPARTMENTAL GROUP/TEAM	PERSONAL
machine	mainframe	mini	micro
price range	500K-5M	10K-500K	1K-10K 10K-50K**
communications	coupled to terminal network	to mainframes, peers and PC's	terminal emulation
database	organization's archives as service	organization, project function	personal files
terminal access	"glass keypunch"	"glass Teletype"	direct CRT
typical uses	corporate accounting, electronic mail	project CAD, order entry, small-business "mainframe"	word processing, financial analysis, CAD**
caretaker	a staff providing service	distributed with user group	the user

Comments (1985):
* Not including regional-styles supercomputers
** Workstations used on a one at a time basis.

from functionally independent modules, one would expect two design approaches:

1. decomposing systems to provide shared LAN services, and
2. aggregating Personal Computer to Form PC Networks and Clusters.

Decomposing Systems to Provide Shared LAN Services

As shared computers become more complex and more centralized, it's desirable to decompose the functions for execution on smaller computers that can be distributed to be nearer the use. Thus, the decomposition of a shared system into various boxes, each of which perform a unique function

permits the evolution of the parts independent of the whole, the physical distribution of a function and the ability of several computers to share a function (see Figure 13b). While we have described the evolution of Local Area Networks (LANs) as a decomposition of a single system, LANs are generally an aggregate of heterogeneous systems which access a shared service of some kind as described below.

LANs differ from Wide Area Networks (WANs) in that they assume a low latency, high bandwidth interconnect. This permits file access as well as file transfer applications. With file access, it is possible to remotely locate part or all of a system's mass storage to a file-serving computer. File access requires bandwidth and latency which are roughly equal to that of a disk (i.e., 10 Mhz rates); file transfer can be done at substantially slower rates (56Khz to 1 Mhz).

Using the reasoning which allowed the formation of the file server, we continue the decomposition of a large central system into servers or stations and then combine these servers into a LAN. The major servers include:

1. Person Server (the personal computer used as a workstation) - local computation and human interface, possibly private storage of files
2. File Server - mass storage
3. Compute Server - batch computation or existance of particular programs
4. Print Server - printing and plotting of graphics images
5. Communication Server - terminal, telephone and PABX, Wide Area Network access including interface to international standards, other companies (eg. SNA)
6. Name/Authentication/Directory Server - naming the network's resources and controlling access to them.

A LAN formed as a complete decomposition of a single system and containing no other incompatible servers would be defined as a homogeneous cluster of Personal Computers or Workstations.

Aggregating Personal Computers to Form PC Networks and Clusters

As personal computers require more facilities (e.g. printing, communication and files), and the number and type of PCs grow, the need to directly communicate for sending messages and sharing files. Furthermore, as a collection of computers in one place forms, economy is gained by sharing common facilities such as printers, phone lines, and disks. Applenet and Corvus Omninet are relatively short and low data-rate Local Area Networks used to permit the construction of what might best be called a network of Personal Computers because of the heterogenity of type. The 3Com system for interconnecting IBM PCs is more characteristic of the homogeneous network, or cluster.

For a PC cluster, one would expect to have a single

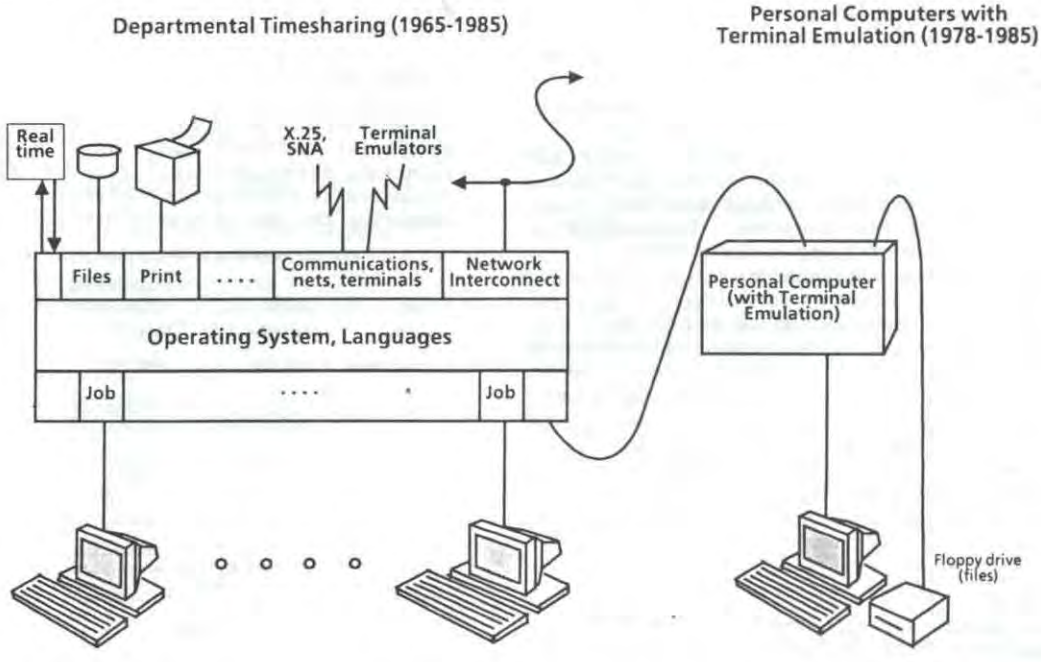


Figure 13a. Departmental timesharing (1965-1985) and personal computers with terminal emulation (1978-1985).

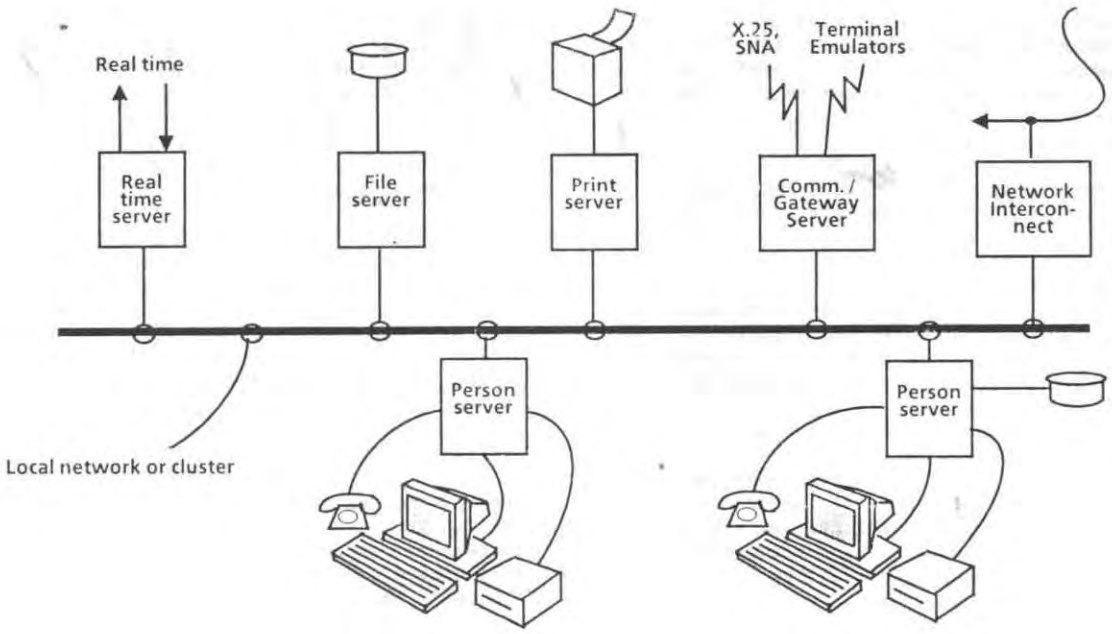


Figure 13b. Personal workstation computing environment using LAN clusters (1980's).

File Server which can supply records at random to any of its constituency. Table 5 summarizes what timesharing, PC's and PC Clusters provide.

DISTRIBUTED PROCESSING USING CAMPUS AND WIDE AREA NETWORKS

The proliferation of timeshared computers required the development of networking in order for various systems to communicate with one another and to mainframes. Thus, dispersed computing became distributed computing. Store and forward wide area networks evolved from the ARPAnet, which was used to interconnect timeshared mainframe computers (mostly PDP-10's).

Table 5 - Timeshared, PC, PC Cluster and PC Network Characteristics (Circa 1981, G.Bell)

WHAT	TIMESHARED SYSTEM	PERSONAL COMPUTER	PC CLUSTER*/ NETWORKS
processing	highest peak	lo-med, guaranteed	= PC
programs size	very high peak	small to medium	= PC
filing	large	small, guaranteed (+ off line)	= PC and TSS
communication	network	term. emulation	= PC and TSS
CRT	slow response "glass Teletype"	fast response, screen oriented	= PC = PC
cost	fixed, can go to lowest$/terminal	lowest entry	f(no. of PCs)
secure	shared, public access	totally private	contained/TSS
pros	explicit costs shared programs, large jobs	low entry cost, "owned" by indiv., security, SW publishing = low cost	ability to expand, shared facilities, better match to org. structure
cons	shared, poor response for terminals, higher entry cost, security	limited capability, but increasing most rapidly	limited proc/prog., shared facilities
heterogenity	one homogeneous system	micro computers	one system/ can be heterogenous

*Personal workstation environment

Campus Area Networks

When a collection of Local Area Networks are connected together in a single area which extends beyond a typical LAN, we call this a Campus. Universities clearly typify the campus as does a collection of buildings. Gateways are used to interconnect LANs of different types (eg, Omninet, Ethernet (802.3), IBM Rings (802.4), Applenet, Arcnet, Broadband token bus (802.2) PCnet), whereas bridges or repeaters are used to interconnect networks of the same type to form one larger network. Bridge technology is necessary in connecting multiple LANs together on a wide-area basis using high-speed links (e.g., satellites).

Wide Area Networks

WANs are characterized by low bandwidth (up to 56 Kbits), high latency, errorful transmission, and autonomous operation of the nodes. The applications typically include: mail, file transfer, database query, and low interaction remote terminal access. Wide Area Networks can be constructed in several ways: direct dial up using conventional circuit switching with voice-grade circuits, an intermediate store and forward network such as Telenet, or a hybrid approach where various worker computers do store and forward switching.

THE VAX, HOMOGENEOUS ENVIRONMENT OR "E"

Although the specific design of the VAX Environment began in December 1978 with the approval of the Corporation, it's origins include:

- the original VAX-11 goals for a 1000:1 range of computers;
- evolution of distributed processing minicomputer networks, in Wide Areas, "Campuses", and Local Areas;
- the appearance of powerful Personal Computers and Local Area Networks, permitting the aggregation of tightly coupled "PC networks and clusters" that provide some of the benefits of timeshared minicomputers and mainframes;

- the ability to aggregate minicomputers and mainframes into multiprocessors (the 78.2) and multicomputer clusters (VAX clusters) that appear to be a "single" system in order to provide higher reliability, higher performance, and incremental performance.

The December '78 statement of the Distributed Computing Environment (Figure 6) and subsequent evolution [shown in brackets] was:

> "Provide a set of homogeneous, distributed-computing-system products based on VAX-11 so a user can interface, store information and compute without re-programming or extra work from the following computer's system sizes and styles:
> - via [a cluster of] large, central (mainframe) computers or network;
> - at a local, shared departmental/group/team (mini) computer [and evolving to a minicomputer with shared network servers];
> - as a single user personal (micro) computer within a terminal [and evolving to PC Clusters];
> - with interfacing to other manufacturer and industry standard information processing systems; and
> - all interconnected via the local area Network Interconnect, NI [i.e., Ethernet] in a single area, and the ability of interconnecting the Local Area Networks (LANs) to form Campus Area and Wide Area Networks."

Figure 7 shows the origin of the "E" shape that characterizes the VAX Homogeneous Computing Environment. The three horizontal segments of the E provide the different computing classes which roughly correspond to different-priced computers; the functions are described in Table 4. In order to implement the environment, many requirements were initially posited, and several developments evolved from necessity:

- a range of VAX-11 and 11-compatible computers to meet the requirements of the various computing styles based on different classes of computers;
- interconnection schemes and the corresponding protocols for building multiprocessors, tightly coupled centralized VAX Clusters, LAN-based PC Clusters, LANs, Campus Area Networks and Wide Area Networks;

GOALS OF THE ENVIRONMENT

PRODUCT RANGE

The important goals and constraints of the Environment are contained in the original statement about what the Environment, which is simply "to provide a very wide range of interconnectable VAX-11 computers". The original goal of VAX was to be able to implement the range [for what appears to be a single system] of a factor of 1000 price range... with no time limit given. Since a given implementation tends to provide, at a maximum, a range of 2-4 in price and 10 in performance if performance is measured as the product of processor speed times memory size, then many models and ways of interconnection were required.

At the time the 780 was introduced, the total range of products for both the VAX-11 and 11 family was almost 500, beginning with $1,000, LSI-11 boards and going to a $500,000 VAX-11/780. The VAX 8600 increased the performance range by another factor of four. If the LSI-11 is included as a personal computer, the price range is reduced to only a factor of 50. While the two ends of the system were "compatible" and could be interconnected via DECnet, they lacked the coherency necessary for a fully homogenous computing environment.

By introducing "VAX Clusters" (Figure 7), the range can be extended by a factor of up to the number in the cluster, or about a factor of 10 more. For VAX, Digital now provides a price range of from about $20,000 for a MicroVAX II to about $7.5 million for a cluster of twelve 8600's and a corresponding performance range of several hundred. The VAX Cluster, shown at the highest level of the hierarchy appears to the user as a single, high availability system.

The VAX Cluster work was initiated concurrent with the VAX Strategy to address the fault-tolerant, and incremental expansion needs for shared computing including a common disk and file system for the cluster. The interconnection of the clusters, using the Computer Interconnect, CI, was the beginning of a standard's activity for interconnecting computers on a standard basis.

STATIC AND DYNAMIC ASSIGNMENT OF PROGRAMS TO NODES

Ideally, a user can decide on how to compute on a completely variable basis at the following times:

- at system purchase or rent time ranging from outside facilities reached via gateways, to a central facility, to a shared department or team computer, to a user's own personal computer.
- at system-use time, ranging from access via a terminal, or personal computer interconnected to the system LAN, or a particular shared computer. Here, work is bound statically to a particular set of system resources. Most likely, particular nodes would execute special programs on data located at the node.
- at task time on the basis of reliability. VAX clusters provide for dynamic allocation of work among the computers in the cluster to effect load balancing since files are centralized.
- at task-use time on a completely dynamic basis, ranging from computing on his own local system to being able to collect any resources and move work dynamically while programs are in execution. With this ability, as a program goes though its various stages of development, it might be moved from small system to large system to take advantage of increased computational power at higher level nodes.
- at task time on a dynamic basis with the ability to acquire arbitrary resources to engage in parallel computation.

CONCLUSIONS

The VAX Homogeneous Computing Environment is an important computer structure for providing a computing environment. This environment encompasses all styles, sizes, and computing classes, from the traditional mainframe and the high availability cluster (utilizing a fault tolerant architecture) to distributed minicomputers, and now the distributed personal workstations. It was stimulated by a number of factors including the distributed-workstation-environment idea. In terms

of architectural scope, it is much broader than simply a range of computers as typified by the evolving IBM 360 -> 370 -> 43xx -> 30xx series because it provides full compatibility through homogeniety for a complete range of computing styles and not requiring separate architectures for shared-minicomputer systems (System 36, System 38, and Series 1), personal computing (IBM PC, word processor) or terminal switching and gateways (e.g. IBM 8100).

Note: this Appendix is solely the view of the author and is not necessarily the view of Digital Equipment Corporation or its engineers.

REFERENCES

Bell, C.G., "The Multi: A New Class of Multiprocessor Computers", Science Vol 228, No. 4698, 26 April 1985, 462-467.

Bell, C.G., H.Burkhardt, S.Emmerich, A.Anzelmo, R.Moore, D.Schanin, I.Nassi, and C.Rupp, "The Encore Continuum: A Complete Distributed workstation-Multprocessor Computing Enviroment", AFIPS Conference Proceeding, 1985, Vol 54, 147-155.

Bell, C.G., G.Fujisaki, J.M.Heinz, A.S.Stevens, and A.S.House, "Reduction of Speech Spectra by Analysis-by-Synthesis Techniques", The Journal of the Acoustical Society of America, Vol 33, No. 12, 1961, pp1725-1736.

Bell, C.G., J.C.Mudge, and J.E.McNamara, Computer Engineering, Digital Press, Bedford, Massachusetts, 1978.

Bell, C.G. and A.Newell, Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971

Burger, R.M., R.K.Caving, W.C.Holton, and L.W.Sumney, "The Impact of ICs on Computer Technology", Computer, Vol. 10, No. 10. October 1984, 88-96.

Clark, W.A. and C.E.Molnar, "The LINC: A Description of the Laboratory Instrument Computer", Annals of the New York Academy of Sciences, Vol 115, July 1964, 653-668.

Metropolis, N., J.Howlett, and R.Gian-Carlo, editors, A History of Computing in the Twentieth Century, Academic Press, New York, 1980.

Pake, G.E., "Research at Xerox PARC: A Founder's Assessment", IEEE Spectrum, Vol. 22, No. 10, October 1985, 54-61.

Perry, T.S., and P.Wallich, "Inside the PARC: The Information Architects", IEEE Spectrum, Vol. 22, No. 10, October 1985, 62-76.

Ralston, A. (Editor) and E.D.Reilly (Assoc. Editor), Encyclopedia of Computer Science and Engineering, Second Edition, Van Nostrand Reinhold Company, 1983, New York.

Redmond, K.C. and T.M.Smith, Project Whirlwind: A History of a Pioneer Computer, Digital Press, Bedford, Mass., 1980.

Roberts, L.G., "Computer Network Development to Achieve Resource Sharing", AFIPS Conference Proceeedings, SJCC, Vol 36, 1970, 543-549.

Siewiorek, D.S., C.G.Bell, and A.Newell, Computer Structures, McGraw-Hill, New York, 1982.

Wexelblat, R.L., editor, History of Programming Languages, Academic Press, New York, 1981.

A PERSONAL VIEW OF THE PERSONAL WORK STATION
SOME FIRSTS IN THE FIFTIES

By

Douglas T. Ross
SofTech/MIT

Section 1

INTRODUCTION

"What the h--- kind of program do you have there?" -- the intercom blasted forth, the sound distorted by the intensity of the shout. It was the chief computer operator in Test Control (as the standard control area for the Whirlwind I Computer was called) calling me in great agitation. I couldn't reply at once, for I was some twelve feet away from the loudspeaker of the E31 console -- in the far back corner of the secret and darkened Room 222 of the Barta Building at MIT. I was hunched over a 16 inch oscilloscope called the Area Discriminator. It was mounted vertically in a box on the floor, so its face made a horizontal flat surface. Adhered to the tip of my moving finger was a bright, glowing displayed spot of blue-white light, about 1/4 inch square. I was in the process of writing my name into the computer with that spot -- freehand. It was sometime in the Fall of 1954.

It was one of the few programs I ever wrote that worked the first time. I remember that I dreamed it up on a long flight back from Texas, which must have been at most a few days earlier. Fewer than 200 instructions were required, and the hardest part was understanding the octal constants that calibrated the scope display coordinates.

1.1 My Sources

This is my second foray into the tribulations of writing (rather than making) history. In 1977 I worked for months on my paper Origins of the APT Language for Automatically Programmed Tools for presentation at the History of Programming Languages Conference June 1-3, 1978 in Los Angeles. (Ross 1977) As was the case for that paper, I have taken as my primary source my extensive personal collection of archival working papers, reports, and records which I have retained over the years, and I have attempted to include only observations I can directly support from these records. Unfortunately I am less sure than was the case for APT, whether or not my current story is complete, however, for the early portions hinge on older material that is less organized and perhaps less complete. I say "perhaps", because I cannot determine whether certain materials I remember seeing are missing, or merely are misplaced. At one point in SofTech's impecunious past, rather than buying needed file cabinets, all of my historic files were without my knowledge summarily crammed into a large number of storage boxes whose length dimension was more than one file drawer. I still have not been able to retrace what was where, as useful chinks in the boxes were filled here and there by segments of unrelated files! In later years I may be able to fill in more details, as relevant portions come to light, but even so, what I have found already yields an engrossing trail.

As in the APT history paper, to provide concise reference to the materials in my files (which ultimately will reside in the MIT Archives) without spelling out each item I have used the following condensed notation: [C540123] means "correspondence 1954 January 23"; [R56123] means "daily resume (a form of personal professional diary) 1956 December 3"; and [WW2Q55p10] means "Whirlwind I Quarterly Report Second Quarter, 1955 -- page 10", see (Project Whirlwind 1952-1957) in the list of of References.

1.2 The Theme

I am not a historian at all, and can't bring myself to write a straightforward (dull) recounting of events. For these stories from the past I feel I must have a theme to provide some perspective. Also, by writing in the first person, I can try to give some of the feel of the times in ways that I hope add interest. But also I think such a style allows a more accurate interpretation of what actually was in the air as the documented events took place, even though

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0019 $00.75

surely all that is written is biased by recollections, as well.

I take the unique event of the first hand-drawn input to a computer as the starting point of my theme for this paper -- that The Personal Work Station as a working reality (rather than a speculative idea) did have a recognizable era of beginning more than thirty years ago. Like the present era, where the physical and performance characteristics of the work station aspect of the PWS are so essential, the fact that Whirlwind had the Area Discriminator scope was an essential ingredient of my successful program. But my theme is much sharper than that, and in a sense even contradicts the natural emphasis on the work station characteristics which dominates most writing on the topic. My real theme is that it is the revolution in personal work (i.e. explicitly the PW, rather than the WS, of the PWS) that was, and is even more so today, what is important. (This also is why, throughout this paper I purposely spell PWS with three words rather than the accepted (correct) two -- to stress my view.)

I hope to show that from the very beginning, the changes in personal working style, made possible by the idea that man and machine can share in the problem-solving process, is what makes the Personal Work Station idea a unique and valuable departure from the customary view of using computers. With the PWS idea, the computer becomes a partner not a mere tool. And to integrate and actually realize this PW and WS idea, requires a systematized treatment of the software that links man to machine. That is the third component of my theme.

1.3 Overview

In the time available to me, I can trace only the beginnings of this theme. As was the case with the APT story, I find that my records show that this PWS story breaks "naturally into a number of overlapping but nonetheless distinct periods during which the focus of attention highlighted first one and then another and then another aspect of the overall subject." (Ross 1977, p. 282.)

The first period covers my introduction to Whirlwind and programming, in the Summer of 1952, and brings out that even with all its rooms full of equipment in its own building at MIT, Whirlwind was a miniscule microcomputer by today's standards for personal work station capacity. And to start with, Whirlwind lacked some needed features, as well.

The second period, from August 1952 through August 1953, saw the flexibility and capacity of Whirlwind expanded, as facilities geared up for the focus on the development of the Cape Cod System -- the R&D base that led to the full-scale development of the SAGE Air Defense System by Lincoln Laboratory. (Redmond and Smith, pp. 201-213.) Most of the environmental equipment for PWS evolution now was present, but my available records don't indicate whether or not I yet had discovered the wonders of the mysterious Room 222.

In any case, well before the Summer of 1954, during the third period when I completed my Masters thesis (Ross 1954b) and our work on Large Scale Data Reduction programs (Ross 1953b) actually got under way (which was the driving force for my own insights and endeavors), I did gain access to the classified areas of the Barta Building and knew that we simply had to incorporate the marvels of manual intervention (or MIV as it was then called) into our system capability. The problem was that I didn't know how to program for all that equipment.

The problem was solved in the fourth period (June 1954 through early 1955) by hiring Bill Wolf from Lincoln Lab. Bill had done some of the early MIV programming for Lincoln. His early departure forced me to learn what I needed to know as I completely rewrote his code to conform to my plans, which were more general than Lincoln's use of the same WS equipment. This also, of course, was the period when I wrote the Scope Input program. (Ross 1954d) Another innovation triggered by my needs was the creation of the Director Tape Utility program [WW3Q54p7] -- the first real Operating System command language system (to use present-day terms) to eliminate the computer operator function for my elaborate, multi-tape runs. By the third quarter of 1954 my plans for MIV use were quite general and well developed.

The fifth period focuses on the preparations for and delivery of two Servo Lab symposia at MIT. The first, held on March 8 and 9, 1955 on "Design and Evaluation of Bomber Fire-Control Systems" (MIT Servo Lab 1955) covered our project's entire hardware and software system for complex system testing. The second, on June 1 and 2, 1955, covered just the MIV-Controlled Data Reduction Methods. (Ward and Ross 1955) Both symposia demonstrated the new techniques to large groups from outside MIT. My PWS ideas were now a working reality, in this particular setting.

The sixth period concerns my first professional paper, Gestalt Programming: A New Concept in Automatic Programming (Ross 1956b) which I presented as the opening paper at the Western Joint Computer Conference in Los Angeles, February 7, 1956. (WJCC 1956) I actually had proposed the paper in November 1955, [C551117, to B.J. Bennett, Prog. Chrm.] and had presented a preliminary version at MIT on January 11, 1956, [C56015] but as I recount here, it was a long process to arrive at a reasonable formulation of the ideas in acceptable form, at the beginning. I consider that paper to be my own definitive statement of the PW theme of this current PWS paper, but written at that earlier time.

The seventh period actually overlaps periods two through six, in time, for it

concerns the long period when, with John Ward and others on the Fire-Control System Evaluation Project, we formulated, proposed, designed, and then assisted in the installation of a full-scale Charactron-based MIV Console for the Univac 1103 computer at Eglin Field Air Force Base in Florida for the evaluation of the B-58 "Hustler" supersonic bomber tail turret. (Ward and Ross 1956) That the actual testing would be done at Eglin had been known since at least early 1954. I had done a special Charactron demo for the June 1955 symposium, and the MIV console itself finally was installed in 1957. [C571024, J.E. Ward to J.L. Moser, Stromberg Carlson] The Whirlwind facilities were duplicated, and their actual circuit drawings were used in the design, thanks to Lincoln Lab cooperation.

Even the Whirlwind setup was not complete as yet, however, and the eighth period mentions briefly my proposal (March 1956) (Ross 1956d) for direct Flexowriter keyboard input to Whirlwind to complete our MIV facilities. Until that time, the Whirlwind Flexowriters had been hooked up and used only for printed (or punched) output and only punched tape input. Barriers to the Gestalt Programming concept, both conceptual and physical, were very high in those days, and even when the Flexo Input was available, its use was a far cry from the central role that keyboards play in today's PWS and word processing practice. We lacked the computing capacity for full on-line keyboard use.

The Flexo Input was just one part of the facilities incorporated into the SLURP System (Servo Lab Utility Routine Programs) for large-scale, experimental program development (Ross 1958) -- which is what the Gestalt Programming concept then had evolved into. It was made possible by the luxury of the expansion of Whirlwind's magnetic core memory to 6K 16-bit words (yes, that's 12K bytes, in today's terminology!) with the cf, change fields, instruction to select any two 1K banks of words at one time. [WW3Q56p44] SLURP was the first complete interactive software development environment, with many programming, language development, debugging, and display tools all integrated into an interpretive environment with Group Control for core vs. drum storage management, and automatic logging and log playback of all MIV actions. Reminiscent of today's debate regarding menu control and how many buttons to put on a mouse, a universal "MIV Box" subroutine gave complete control over these hierarchic facilities with only Major and Minor Exit Buttons and one Wait switch for stepping and selecting from one control level to another. From today's vantage point, I can't imagine how we did all that in just 12K bytes with only a few (2 or 3?) times that much magnetic drum backup. But we did. I take SLURP to be the systematized solution, third component of my PWS theme in this paper. Development continued until the demise of Whirlwind in 1958.

The ninth period artificially cuts off abruptly at the start of 1960, to match my title and put a stop to burgeoning growth of what might be covered. Its beginning overlaps the other periods of focus, with the plans and actions to carry on MIV work on the IBM 704, 709, 7090, and 7094 computers at MIT, after Whirlwind's demise. Although artificial, this actually is a good stopping place, because other, broader themes spring from this period. Timesharing, computer graphics (both hardware and software), computer-aided design, software engineering (both language and practice), and software technology tools and methods all blossomed and flourished at MIT in the 1960s, and have direct ties to the story I present here. But those are tales for another time.

Section 2

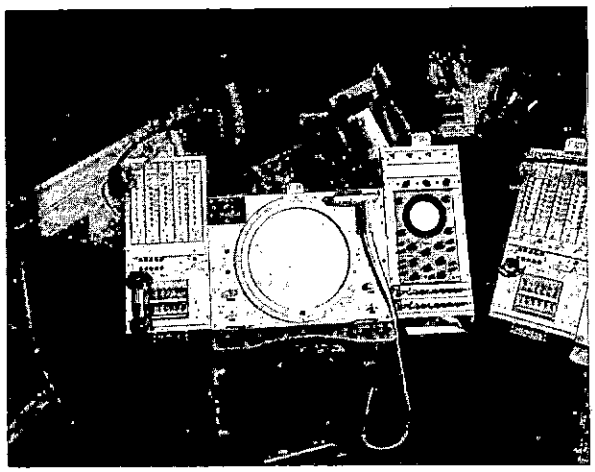
PREPARATION

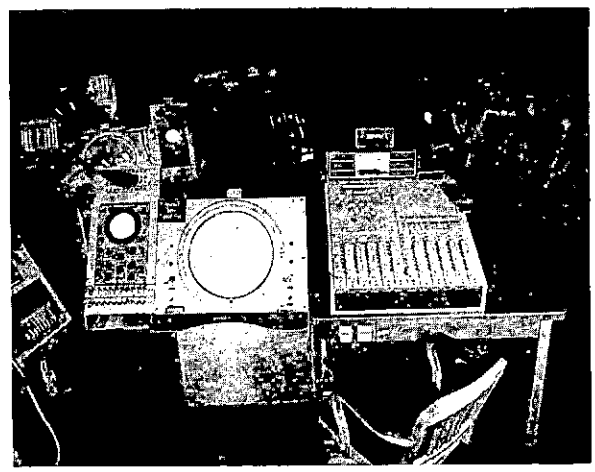
Significant events always are the rearrangement into new forms of otherwise ordinary and insignificant, routine happenings. The ordinariness allows the significance to be recognized. What was the environment, and what was going on at Whirlwind that triggered a new and more intimate kind of computer use? What were the pieces from which the PWS idea began in this story? The preparation for my theme also is the preparation stage of my story.

2.1 Programming Background

I came to MIT in the Fall of 1951 as a Teaching Assistant in the Mathematics Department, having received an AB Cum Laude in Math from Oberlin College in June. Pat and I had been married the previous January, and she was the first of a group of "computers" hired by Lincoln Laboratory (Badge No. 161) which was just getting under way -- performing calculation and graphing assignments for the engineers, using Marchant desk calculators. One of her assignments involved using a mechanical correlation computer, designed by Norbert Weiner, which was in use in the Servomechanisms Laboratory at MIT. (Ward 1954) When I needed a summer job for 1952, I applied to the Servo Lab, and was hired also as a "computer". One of my first jobs involved checking some anomalous points that had been obtained on that mechanical correlator, and someone suggested that I see if Whirlwind could be used, for they knew that Jack Arnow had written such a program. Jack later founded Interactive Data Corporation (now part of Chase Manhattan Bank) as one of the early timesharing service companies.

I, of course, had never heard of Whirlwind or of real computers, so I was fascinated. I still have Jack's little program, plus the Digital Computer Lab application form for Problem #87 (July 2, 1952) to get computer time, some scribbled yellow papers where John Frankovich (still working at Lincoln Lab) gave me my first programming pointers, and the "Tape 1414 Mod 0" July 11 printout of my Auto-correlation program. My earliest Run Request (July 18) shows that I was already up to Mod 3 one week later. (Ross 1952)





(Left) Cape Cod Console with light gun and Manual Intervention (MIV) equipment. (Right) E31 Console in Room 222. Note intervention register octal keyboard banks and intervention switches, activate pushbuttons and lights, functionally arranged. Intercom and computer restart button on right.

Before I could complete the debugging of my rather elaborate program (which later was used in many projects and theses by others) Whirlwind was completely shutdown from August 11 through August 30, 1952! [WW3Q52p6] I used the time to write a matching Fourier Transform program so that power density spectra could be computed (which also was widely used, and formed the basis for an improved program for my own Master's thesis in 1954). The reason for the shutdown is the real starting point for my PWS theme -- for in that period the entire input/output system for Whirlwind was drastically modified to provide the elaborate hardware facilities needed for Whirlwind's new and primary Lincoln Lab mission -- high priority development of an air defense system for the United States and North America.

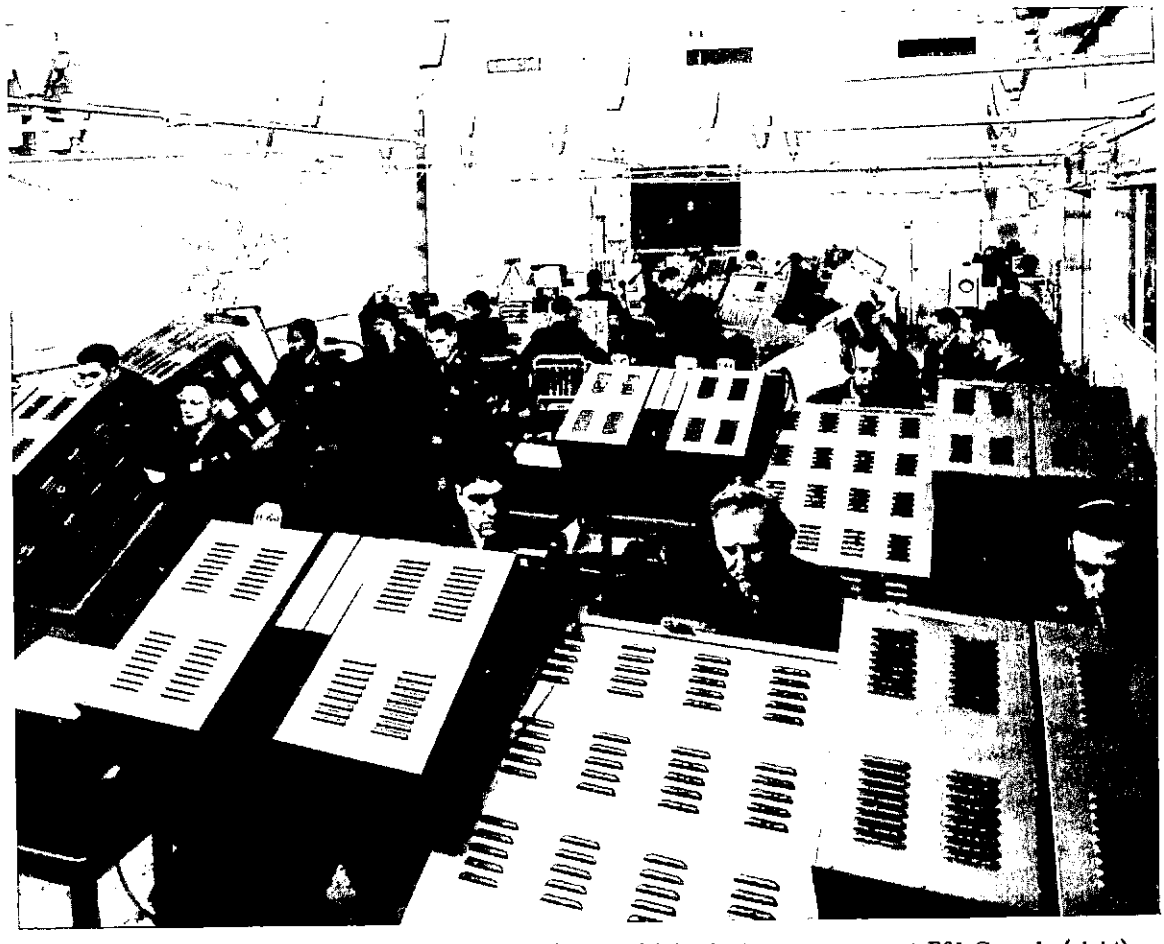
The Whirlwind order code (of 32 instruction types, maximum) was revised so that si, so, rd, rc, bi, and bo for "select input", "select output", "read", "record", "block in", and "block out" could be used for any one of a large number of input/output lines to which analog scopes, mechanical switches, "activate" (one-shot) pushbuttons, indicator lights, and magnetic drums could be attached. The mechanisms for general manual intervention in the running of programs were at hand.

Only the barest essentials trickled out to the programmers of the user problems. Most of the subsequent construction took place behind the locked, green, double doors of Room 222, which was strictly off limits. With the end of the summer at hand, and with my important programs in limbo and my enthusiasm for programming running at white heat, my supervisor, John Ward (still at MIT) asked me whether I really wanted to go back to teaching Freshman Calculus, or would I rather make a career for myself in Servo Lab, full time, taking courses as a Special Student. I felt obligated to fulfill my 1952 Fall teaching assignment, so we arranged for me to do that in addition to working full time programming and taking graduate math courses for that semester. The Math Department wasn't happy to loose me. I had been so serious and successful about by Teaching Assistant role that I had been appointed as the only graduate student member of the Dean's group of Freshman Advisors. But I remained with the Lab until August 1969, when my colleagues and I left to found SofTech.

2.2 Project Background

The Air Force-sponsored project headed by John Ward was concerned with the test and evaluation of the accuracy and performance of airborne fire-control systems -- specifically servo-controlled tail turrets in bombers. The autocorrelation and Fourier transform programs were part of the study of radar noise analysis. I was the only mathematician in the Laboratory, and one reason for hiring me was that a new form of Air-Mass Ballistic Tables had just been developed, (Nielsen and Heyda 1951, p. 9) more accurate than the old tables which had been developed for ground artillery and required extensive compensation to account for rapid movement through rarified air. My first major assignment (November 1952) was to see how these new tables could be used for the evaluation problem.

By January 12, 1953 I had a complete data flow diagram (Ross 1953a) for hit probability density calculation, based on fourteen measured inputs, and using vector calculations well suited to the three-dimensional problem and



Room 222 of Cape Cod System. Closest two airmen with backs to camera are at E31 Console (right) and adjacent console (left) used for Gestalt Programming. Directly beyond them is tripod and box of the Area Discriminator light cannon. Originally walls were black. Indirect ceiling lamps had red bulbs for system operation. (2/19/54)

digital computing. With further elaboration and refinement over the next five years, the complexity of actually programming and operating this solution for production data reduction (Ross 1953b) was the stimulus for all my MIV developments until the advent of the Computer-Aided Design Project in 1959.

The main part of the project was directed toward the development of airborne flight test instrumentation that could measure the many variables during mock attacks by a fighter aircraft pursuing a bomber. With the feasibility of the digital computer data reduction established, attention focused on digital instrumentation (primarily shaft encoders to convert selsyn-repeated analog quantities into Gray-coded binary) for recording on a flyable magnetic tape that then could automatically be read into the data reduction computer on the ground. (Telemetry was too unreliable in those days to risk on such an expensive program.) By 1954 our efforts were targeted toward meeting the needs for evaluation of the XMD-7 Fire Control System being built by Emerson Electric for the secret supersonic B-58 "Hustler" Bomber. The B-58 was not yet ready, so the program was called "Pre-B-58", and tests were run with the B-47 in the beginning. (MIT Servo Lab 1958) It was the most sophisticated test program of its time, and cost over $10,000 per flight hour, just for the aircraft, I believe.

By the Spring of 1953, programming was complete for the autocorrelation functions and work had begun on a Polynomial Fit program (Ross 1954c) to place the ballistics tables in computable form. By the summer, an elaborate Mistake Diagnosis Routine (MDR), (Ross 1953c) which allowed breakpoints to be set in an arbitrary program, with intermediate results saved for printout or displayed on the camera scope, was written and disclosed that the interpretive floating point arithmetic routines of the Whirlwind Comprehensive System (CS) were inaccurate, so they were repaired, to everybody's benefit. [WW4Q53p25andp11] The MDR played a continuing role in our experimental programming environment, especially when it later was placed under MIV control. In the Summer of 1953 the first programming for the

evaluation equations also got underway, [WW3Q53p25] but most of my time in early 1954 was focussed on my Master's thesis on minimizing Gibbs' phenomenon oscillations in the computation of Fourier transforms. But by the Summer of 1954 the programming stage was set for our first serious use of manual intervention techniques in our work.

2.3 Hardware Background

To complete the stage-setting for the Scope Input coup, I will give the following synposis of Whirlwind hardware developments during the corresponding period. In the Summer of 1952, when I started to program on Whirlwind, it had only 1024 words of electrostatic storage (ES) of 16 bits each, and operated at the then-blinding speed of 20,000 operations per second (hence its name). [WW3Q52p4] In those days we used both audio and visual aids in the debugging of programs. Originally, a scope in Test Control duplicated the X-Y deflections of the ES tubes' storage matrix so that the loop structure of programs could be seen, in action. The more often a loop body was executed, the brighter its trace. The audio aid consisted of a loudspeaker attached to a digit (I believe was digit 13 of the Accumulator) so that every time its binary state changed from a 1 to a 0 or back, a beep was heard. The execution rate of program instructions was such that loops of different lengths made tweedly tones up and down the scale, so that you could hear the various phases of the program taking place.

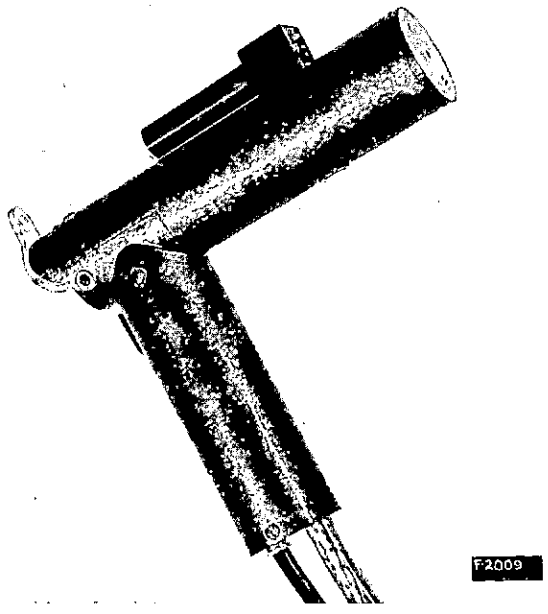
After the shutdown and changeover to the new I/O system in August, 1952, things remained stable for a time, but in the second quarter of 1953 ES was doubled to 2048 words by the addition of another bank of 17 tubes (one was a parity digit). [WW2Q53p30] Then suddenly and smoothly, in the third quarter of 1953, both ES banks were permanently replaced by the first two banks of magnetic core memory (on-line August 8 and September 5, respectively). [WW3Q53p32-34] The magnetic memory cycled at 9 microseconds versus 23 microseconds for ES, so Whirlwind now could do over 40,000 instructions per second. [WW4Q54p4] But still most loop structures remained in audio range, so the loudspeaker retained its usefulness.

The ES scope trace had been abandoned in the fourth quarter of 1952 when my first-ever program for computing the Autocorrelation function couldn't be run without causing Whirlwind to break down. Other programs sometimes gave trouble, too, but mine showed the problem to be that its tightest calculating loop fell in a group of registers which happened to fall in one corner of the square matrix of storage locations. The spherical pattern of the ES tube's holding gun (which uniformly, but with inverse square law fall-off, sprayed the entire storage surface with electrons to compensate for leakage) couldn't be adjusted for the corners without upsetting the center of the matrix, when adjacent bits were accessed so rapidly. The engineers solved the problem by changing the diode pattern of the X-Y decoders so that consecutive words were randomly scattered around the square matrix, so that even tight loops were geographically dispersed. [WW4Q52p6] My program, with a short data tape, was added to the regular maintenance test suite used for ES adjustment and checkout, and the ES memory trace scope was no longer useful for debugging, since loops didn't show clearly.

In the meantime, however, many Whirlwind programs made use of the 16 inch display scopes, and Test Control had two of them mounted high up on a rack. The lower one had a 35 mm camera under computer control, for making photographic records, and the upper one could have all 16 analog display lines ganged together to serve as a monitor check on whatever program was running. Thus the normal operational milieu for the computer operators still had the full audio/visual complex of lights, scope, and loudspeaker. When a strange sound pattern occurred, the reflex action was to glance at the monitor scope, and then start to examine the indicator lights to gauge roughly by brightness where the program was operating in memory or was the accumulator stuck in some repeating pattern, etc. That's what the operators had done, when I got that loud, panic call over the intercom.

There were more than a dozen other scopes in the cavernous black-painted, shrouded Room 222, eerily lit by dark red light. All except the Area Discriminator scope were mounted at a slant, surrounded by banks of push buttons and indicator lights in several styles of interactive consoles for the various personnel needed to perform the many functions of airplane tracking and control of the air defense mission. The most elaborate was the E31 Console of the Combat Data Director, which in addition to its elaborate set of buttons and lights, duplicated the essential master controls for Whirlwind, so that the entire complex could be controlled from that station, bypassing the normal operations of the master Test Control in the main computer room. But I, of course, had no knowledge of any of this, yet -- it was all very hush hush, and you needed special security clearance and a need to know to even take a peek.

My records don't show when it happened, but sometime in 1953 or early 1954 arrangements finally were made with Lincoln Lab to allow John Ward and me to be introduced to the enticing Room 222. I can still remember that first walk through the black-cloth-draped entryway into the red-lit, darkened room just filled with green-glowing scopes and orange-red neon indicator lights. Neat! Just what we could use to obtain mastery over the behemoth Evaluation Program, for whose complexity I already had a growing respect. The functioning of the various parts of the Cape Cod System were explained to us (although it was only later that we saw an actual test run with planes being tracked by the operators, etc.). It was clear that the E31 console, with part of an adjacent console, would be the best control station (PWS!) for general purposes, since the full complex of capabilities (including light gun and more buttons on an adjacent console) was available there along with



Final design of Cape Cod light gun, with sight. (10/2/53).

the duplicated computer controls and the intercom.

The Area Discriminator almost got lost in the shuffle. I now will give a complete description of it. All the other scopes were mounted in consoles with switches, buttons, and light gun for manual intervention. The Whirlwind light gun was shaped like a backwards pistol, with a sight close to your trigger finger knuckle and a wire coming out of a barrel which extended back over your hand. The barrel contained a photo-multipler tube, and the wire connected to an **si** line of the computer. If a displayed spot was in the sight when the trigger was pulled, that would set an activate bit (one-time, read-only) so that a suitable **si, rd** sequence following the **si, rc** that displayed the spot (but before any other spot was displayed) said that the operator had selected that spot from all the others in the display. This was used to assign tracking functions to radar blips which were returns from airplanes.

The Area Discriminator had no buttons, lights, or anything for human intervention. It simply sat in a large box in the back corner of the room. Mounted on the box was a tripod which supported a smaller box with no bottom, so that it was suspended about 18 inches over the horizontal scope face surface. This was the Light Cannon -- a photomultiplier tube mounted so that it viewed the whole scope face and could be **si, rd** read for any spot in the display. When we saw it in the Cape Cod System use, a large circular piece of dark yellow plexiglass had been laid down, centered over the scope face so that only a narrow annular ring around the edge of the scope was open to the Light Cannon's view. The yellow filtered out the blue phosphor flash of any displayed points, so the Cannon was blind to all central display activity, but any fresh radar tracks crossing the perimeter could be seen and could be brought to the attention of the Track Initiator/Monitor operator for light gun tracking. This is why it was called "the Area Discriminator", and with the plexiglass disk in place, it actually was an analog computing element component of the Cape Cod System.

2.4 Software Background

My guess is that our tour of Room 222 probably was sometime in the Spring of 1954, for the three 16" scopes mentioned in the fourth quarter 1952 Whirlwind Quarterly Report (at the time of the I/O shutdown) had grown to more than twenty, and I think all the Cape Cod consoles were complete at the time. In any case, I had my thesis to finish, Dick Turyn who had been programming on the Evaluation Program was leaving so I needed a replacement, and we needed to learn the ins and outs of Room 222 programming. Therefore we hired Bill Wolf from Lincoln Lab, solving several problems at once. He came in June 1954, but stayed only about six months, (MIT Servo Lab 1952-1958) when he left to go into consulting for himself. Later he formed Wolf Research and Development Corp., which acquired Whirlwind itself, under Navy sponsorship, when Lincoln and MIT had no further use for it. (Redmond and Smith 1980, p. 224) Wolf R&D also had the first facilities contract to run NASA Houston computers, I believe.

I guess I'd have to say that Bill was a better businessman than programmer. He did a fine job for me, writing a basic set of programs to use all the functions of the E31 console and a program to display large, fat letters on the scope, big enough to be directly read from the 35 mm film so that we could frame our classified results with "CONFIDENTIAL - SERVO LAB - CONFIDENTIAL" before and after our plots and displays. (Wolf 1954) But the programming style was very tangled and hard to follow (and I suppose not atypical of the code of other Lincoln people who were learning to master the tricks of this new manual intervention trade, for that was a major reason I hired him). Bill's code was far from debugged when he got the wanderlust. I hadn't been working closely enough with him to know what I was getting into, but I said I'd finish the debugging because that would give me the education I'd wanted anyhow. So off he went to make his fortune.

Almost immediately I saw that I would have to completely rewrite and replace Bill's work, and proceeded to do so. Many features of modularity, flexibility, robustness, and generality were needed, in order to match my growing vision of what general MIV (i.e. the PW functionality of PWS) was to be all about. Without Bill's efforts to critique, many of these aspects (which I certainly had not articulated beforehand as specifications) would not have occurred to me until much later. Also some basic principles of MIV programming (that I have seen violated repeatedly in the '70s and '80s as the microcomputer age programmers have

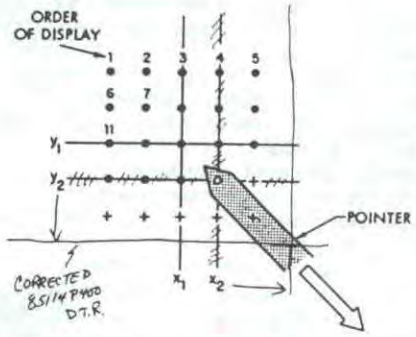
re-encountered and reinvented those 30-year old phenomena) showed up in my critique of Bill's code. For example, he had the **si, rd** to read a switch directly at the point-of-use in the program flow, throughout. I structured separate read cycles and stored the switch settings in program variables. Not only did this make the overall program better structured, but it also meant that anything that could be done manually also could be done automatically, because actual program control came from the stored variables, not from the switches themselves. This feature made possible our log and playback of manual actions as well as test simulations.

2.5 Director Tapes

It was while I was first reading, with some understanding, about the complete set of **si** capabilities of Whirlwind (including the Lincoln Lab-only portions, previously unknown to me) that I discovered that the mechanical tape reader of the Test Control Flexowriter was still operable, even though there always had been a photoelectric tape reader (PETR) since I had been around. Although the print mechanism of the Flexowriter had been in daily use all along, for operator communication, and the direct tape punch recorded some logging information, the tape reader had no current use. The number of separate tapes and the complexity of operator instructions for our Evaluation Program runs was getting so bad that I suggested to John Frankovich and Frank Helwig (on the Digital Computer Lab staff) that the Whirlwind utilities should be expanded to put all these instructions on a tape that the mechanical reader could read, while the main tapes were read in by the much faster PETR. It was worth the wait for that idea to get worked into their schedule, [WW3Q54p7] for the Director Tape program, as it was called, included not only the terse control language basics, but also conventions for labelling punched tape headers as to type (binary, program, post mortem request, etc.). When Director Tapes were further upgraded in the Summer of 1956, they still were not used by many programmers (the tape room didn't like to splice all the tapes together). [WW3Q56p23] But they were essential to our Room 222 runs, for they allowed us to completely eliminate the operators. I believe the Director Tape capability was the first true Operating System in the modern sense of the word, especially when combined with our beginning versions of Group Control (based on the earlier MDR needs) [WW4Q54p15] for manipulating drum storage. Director Tapes and Group Control are parts of the systematized solution third component of my PWS theme.

2.6 The Scope Input Program

My records don't show exactly, but I do remember dreaming up the Scope Input program on a long flight back to Boston, and my records show that I did attend an early Pre-B-58 Meeting with John Brean (the lead engineer on the Project's instrumentation work) in Dallas, on August 9, 1954, so I'd guess it was about that time. (Convair 1954 and [C54817 to S.C. Marcus, Emerson Electric]) Bill Wolf still was trying to really get his programming under way, but various test runs were being made by both him and me in Room 222. So I knew just enough to write the program. Perhaps triggered by the Dallas discussion about all the things that could go wrong with getting flight data from the plane into the computer via a crude tape system, I envisioned patching up bad spots in a plot of the input data, free hand. We already had had enough experience with the Polynomial Fit and a Lagrange Interpolation programs that had been written in preparation for the Evaluation Program, that I didn't trust any analytic methods. As the earliest written reference I have found so far (Fourth Quarter 1954) says: "A 'scope input' routine has been written which gives a type of two-dimensional analog input to the Whirlwind computer. The principle of the program is similar to the 'flying spot scanners' used on analog computers. The equipment used is a 16-inch oscilloscope under the control of the computer with a photocell mounted so that its field of view is the entire scope face. Then by programming a flying spot and asking whether or not the photocell 'saw' the spot, the program can be made to follow an opaque pointer as it is moved in a random fashion over the face of the scope. Since the program displays the spot by digital coordinates, the tracking of the pointer constitutes analog input to the computer." [WW4Q54p15]



Tracking principle of first Scope Input program August, 1954. Shadow was tracked using Area Discriminator Light Cannon. (Error correction November 1985!)

The Scope Input program is quite simple and direct. The tracking function itself consists of a tight loop that displays a 5 x 5 square array of close-spaced dots, left to right, top to bottom, interrogating the light cannon after the display of each dot. If a dot is not seen, the array is recentered on that location and the cycle is restarted from the top left corner again. Otherwise the next dot in sequence is displayed and tested. The cycle is so fast that the array can track even quite fast motions of an opaque pointer -- catching up to a new position before it can get very far.

Modified parameters of the same logic allow the pointer position to be located initially. A wide-spaced 32 x 32 array covering the entire scope face zeros in by successive

half-spacing until the tracking array can take over. Since the pointer is hand held, at least the hand will be caught by the initial spacing. This two-mode program was the full extent of the initial Scope Input program, for the intent was merely to demonstrate the tracking principle.

On that fateful afternoon, I set up the Scope Input tape in the PETR, bid the operators adieu without saying anything special, and went off to Room 222 by myself. As I approached the E31 console in the middle of the room, about to lay down my papers on its narrow desk I gasped. The restart button was there on the console, but the Area Discriminator was 12 feet away in the corner! Would I have to call for operator help after all? Suddenly it dawned on me! The top-to-bottom, left-to-right logic of the scan meant that my tracking cycle actually was a maximum-seeker-from-the-left! In other words, whatever shape made the shadow, my spot would rush up its left edge as fast as possible -- and then (by the logic) would get stuck at the very top, because it couldn't track downhill! I saw that that was why I expected it to track at all.

With that insight, my problem was solved. All of our work was done on pads of yellow paper (opaque to the Ozalid copying process). I merely tore off a blank sheet as I went over to the Area Discriminator to remove the plexiglass disk. In its place, I laid down my sheet of paper, roughly at a 45 degree angle. I went back to the E31 console and pressed the restart button. The lights indicated that Whirlwind was running. Was my program in a loop? You bet it was. As I approached the Area Discriminator I could see something was happening. There on the top corner of that piece of paper was a bright spot. The loop was my display loop. So far so good.

Very cautiously I snuck the index finger of my right hand up the yellow page, staying in its shadow. Then very gently I used my left hand to slip the paper out from underneath -- and sure enough, the spot remained -- but now stuck on my finger. I started to write. That's what got the operator's attention in the control room. The almost-random tracking recycling made a rasping sound on the loudspeaker, and that's what first got their attention. The rest is history.

I haven't found the original Scope Input program in my papers, but I do have the version used in the Symposia the following year. (Ross 1954d) (I now realize that I could have simplified the program still further, by using the left-edge-following feature to switch into tracking immediately from the first start-up dot, not seen. Then both the half-spacing logic and the initialization for full 32 x 32 spacing could have been eliminated.) Later I had our model shop make an automatic remote button-pusher, since I wasn't allowed to tamper with any of the Whirlwind wiring. A hand-held pushbutton on a long cord activated a solenoid mounted on a bracket held in place by a loosened framing screw of the E31 console frame, so that the Restart button could be pushed remotely from the Area Discriminator area. A wheeled dolly held a power supply for the solenoid, and plugged into a 110 volt outlet. It worked, and our delicate and unique relationship with Lincoln Lab regarding use of the equipment was undisturbed. All I had to negotiate was a place to park the dolly between runs!

We never got around to actually using the Scope Input program for anything, however. I didn't need it for patching our Whirlwind test data for the Evaluation Program, and for some unknown reason we didn't specify a light cannon for the Eglin Field 1103 Charactron console. We did make a light cannon for the IBM 704 and 709 computers at MIT, but again it only was used for a study of various light pen and other tracking studies in 1960, for the Computer-Aided Design Project. (Ross and Ward 1961) I never was successful in interesting anyone to make a generalized shape reader, which could easily be constructed just by changing the sequencing of the 5 x 5 tracking array (to seek minimum from the right, etc.). So in its historical setting, my Scope Input program was merely an interesting demonstration piece that the crowds loved, but was a bit before its time.

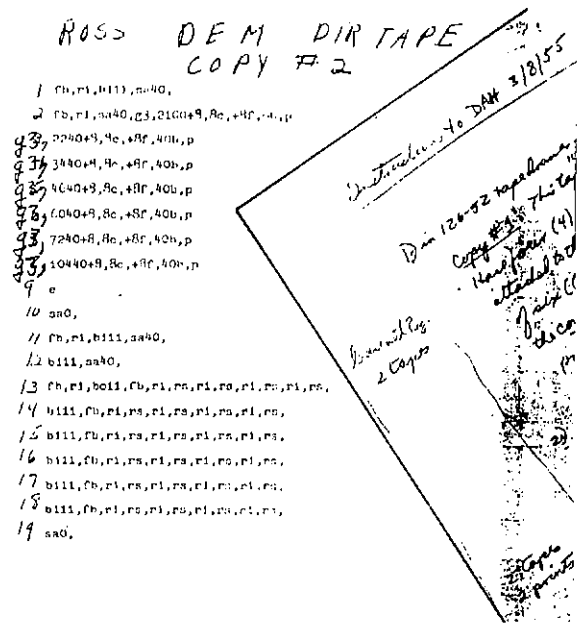
Section 3

PRACTICE

The pieces all now were in place. I could actually practice the new MIV techniques in conjunction with the Evaluation Program. I also could practice talking about what was new and exciting about this intimate coupling of man and machine.

3.1 The First Symposium (March 1955)

In any project, a point is reached when it is necessary and appropriate to demonstrate some initial results to the outside world. This point was reached for the Fire Control System Evaluation Project in the Spring of 1955. Even though my part of the project, on Data Reduction, could have used a bit more time to digest the integration of my reworking of the MIV programming with the parts of the Evaluation Program, a symposium on the Design and Evaluation of Bomber Fire-Control Systems was scheduled for March 8 and 9, 1955. (MIT Servo Lab 1955) The first day covered various engineering matters, mostly concerned with the servo dynamics of a new hydraulic antenna drive, and ended with demonstration of the drive, and (for added interest) the numerically controlled milling machine in Servo Lab. The second day covered evaluation. John Ward presented our philosophy of evaluation, John Brean the concept of digital instrumentation, and I presented the air-mass-based evaluation analysis and MIV-based data reduction programming. In the afternoon, buses shuttled two groups of about 75 people between Servo Lab and Barta Building for digital instrumentation and Room 222 demonstrations, respectively. The Visitors List shows over 170 names from 48 government, industry, and MIT organizations. It was an impressive show.



Printout of ROSS DEMO Director Tape for March 1955 Symposium. Eliminated computer operator actions and presaged Operating Systems of today.

Other than an Agenda and the Visitor's List, there seems to have been no handouts for the symposium. I have a draft copy for some glass slides, an outline of my talk, and a cardboard cutout of a fighter aircraft profile which was placed on the Area Discriminator scope, so the light cannon could talley hits on the target according to the calculated projectile position. (We used actual test data from Project Hornet, an earlier test program carried out by Emerson Electric for the B-52.)

The complexity of the Data Reduction show is indicated by my detailed instructions (dated 3/8/55 for the demo the next day!) for preparing the ROSS DEM. Director Tape. (Ross 1955a) Although I no longer can decipher the director tape language, there are well over 100 operations encoded -- loading tape segments onto the drum, executing initialization sections, calling up individual programs and executing them -- all of which would have had to be carried out flawlessly by the operator, without the director tape feature. (As a matter of fact, I had pre-loaded my paper tapes onto one of Whirlwind's magnetic tape units, for director-tape-controlled loading. The first time it worked fine, but for the second group of people the magnetic tape unit broke down! I handed the microphone to John Ward to fill the gap as best as he could, ran downstairs to the tape room files, rushed back with an armful of disjointed tapes in boxes, and with the help of the operator (probably Joe Thompson, who was the best!) managed to complete the second demo, breathlessly but in full. So we did it both ways!)

Every scope in every console in Room 222 duplicated the displays I was demonstrating at the E31 console, so everybody had a good view. On request, pertinent variables could be monitored in numerical form, the geometry of the encounter could be shown (including firing pattern, measured through the airplane cutout for the Light Cannon), and elaborate, calibrated and labelled plots of all functions could be selected. I also showed the Scope Input capability, as well, The actual demo lasted about half an hour.

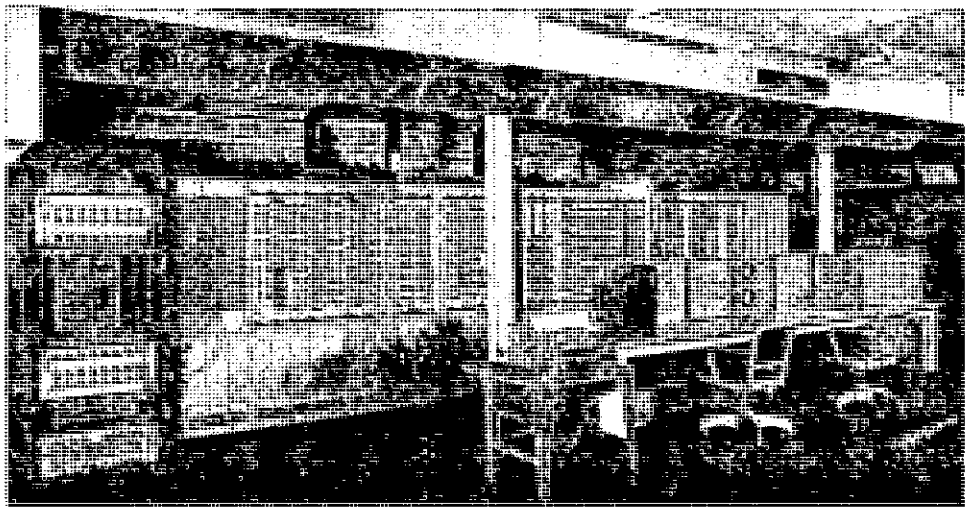
Another interesting tidbit about the demonstration is the following quote from the First Quarter 1955 Whirlwind Progress Report: [WW1Q55p56] "With the present logic the program demonstrated at the symposium had only five registers unused out of a total of about 3,500 actual program instructions. The new logic will be limited only by the drum capacity of the computer and will be more efficient as well." This documents what I remember as a previous panic point in the preparations. The final assignment of Group Control drum and core addresses came out to be almost 40 registers too big! Only by changing the words labelling the scope plots to abbreviations ("AZ" for "AZIMUTH", etc.) was I able to achieve that tight fit in the nick of time!

3.2 The MTC Charactron Demo

On April 14-15, 1955 I attended an Instrumentation Symposium at Patrick Air Force Base, Cocoa (Cape Canaveral), Florida, put on by the American Ordinance Association. (AOA 1955) The most memorable thing about that meeting was the unauthorized detour the on-base shuttle bus took to drive past a large, flat, open area with some raw concrete in the ground, with reinforcing rods jutting skyward -- all with no comment. It was the then-very-secret first launching pad for the Atlas missile, just being built.

The response to the MIV-controlled data reduction in our own March symposium had been so great that we were requested to put on another show on just that topic. On May 2, 1955 John Ward issued an invitation to a Symposium on Data Reduction to beheld at MIT, June 1 and 2, 1955. (Ward 1955) Inspired by my Cape Canaveral trip, and to provide, in addition to the Room 222 demonstrations, a demonstration of the Charactron display tube capabilities, I quickly learned how to program the Memory Test Computer (MTC) at Lincoln Lab. (Redmond and Smith 1980, p. 206) This was a general purpose vacuum tube computer that had been built specially to test the original Forrester magnetic core memory stacks. When the tests were successful and the core stacks were suddenly moved to Whirlwind, another stack was made for MTC, and it was then being used to shake down the Charactron display for use in the SAGE system.

I still have my MTC working papers for my Missile Launch Simulator program, along with the original CS Tape Preparation Requisition for "Tape 0", submitted at "4:25 PM, May 24". (Ross



Memory Test Computer (MTC) with Magnetic Core stack. (4/6/53) Used for Charactron demo in March and June 1955 Symposia on MIV.

1954a) In addition to showing examples of how the MIV monitoring displays of intermediate results would look in Charactron character display format (rather than Whirlwind's elements of a 7-stick figure eight character display [WW2Q53p31]), the program used the vector plotting capabilities of the Charactron to draw a map of the Florida coast plus some islands, and then cycled through a moving display of a missile launch, showing where the pieces would fall if the test director had to abort the mission and blow up the missile in flight. The islanders would have appreciated such thoughtfulness, I reasoned.

3.3 The Second Symposium (June 1955)

Three days later (and only one week before the symposium -- things moved fast, in those days), John Ward invited Pat Youtz (who had led all the Whirlwind work on the ES memory tubes, and now was in charge of SAGE display work) to "describe to the people at our data reduction symposium just what can be done in the near future in the way of input-output." His letter goes on:

"As you may know we are doing data reduction studies for the Air Force and have developed techniques on Whirlwind using the scope and intervention that are a great improvement over any other techniques in use at present. We are building digital instrumentation for a forthcoming field test which will be evaluated on the ERA 1103 at Eglin Field, Florida, and the primary purpose of this meeting is to get the Eglin Field people and the ERA people together and show them what we have accomplished with Bob Weiser's attachments to Whirlwind, with the hope that they will get inspired to obtain similar equipment. We would therefore like to put on as good a show as possible and that is why we would like to show the charactron tubes in operation. We have permission to use the MTC and Doug Ross has written a special display program to put the charactron through its paces.

"It seems to us that if any equipment design is undertaken by ERA that the charactron should be used as the basis for design, and we would greatly appreciate any help you or your group could give us in showing our visitors what the charactron looks like and perhaps your manufacturing and test operation. We have taken the liberty of tentatively scheduling this material on Thursday, June second, as an adjunct to the MTC demonstration at 2:00 p.m. I am sending this to you so that you can think about the business a little and I will call you Tuesday to see if we can make some final arrangements. We would be most grateful for any assistance you could give us in convincing our visitors of the current feasibility of charactron type displays. I am enclosing a brief list of the organizations who will be present at the symposium." [C55527 JEW to Youtz] Pat did indeed pitch in, and both the Charactron and the Memotron (a smaller storage tube version) were included in the June 2, Lincoln Lab visit.

On July 29, 1955 John Ward issued "Minutes of M.I.T. Data Reduction Symposium, June 1 and 2, 1955", Report No. 7138-S-1, (Ward and Ross 1955) which includes my 7138-M-112 memorandum "Special In-Out Equipment for Human Participation in High-Speed Computer Operation", July 25, 1955 (Ross 1955c) -- in which I summarized the "Program Operation" and gave a "Description of Equipment" for Activate Buttons, Intervention Switches, Indicator Lights, and Audible Alarms, each with a diagram, Description of Operation, and "Things to Notice" about how they would be used with the EF (external function) instruction of the ERA 1103.

For the record, John Ward's letter to Youtz shows that the following organizations

were invited to the symposium, and the minutes show how many people actually attended:

Military Physics Research Laboratory,
University of Texas (2)
Emerson Electric Company, St. Louis (3)
Air Force Armament Center,
Eglin Field, Florida (4)
Armament Laboratory,
Wright Air Development Center (4)
Engineering Research Associates,
Remington Rand, Inc., St. Paul (2)
Patrick Air Force Base,
Air Force Missile Test Center, FL (3)
Holloman Air Force Base,
Alamogorda, New Mexico (0)
Air Force Flight Test Center,
Edwards AFB, California (1)
NOTS, Inyokern (0)
Bell Telephone Labs (1)
Convair (3)
Davies (0) [manufacturer of the airborne mag tape unit]
National Security Agency (2)
Watertown Arsenal (0)
White Sands Proving Ground (1)
WCRRU, Aero. Res. Lab. WADC (2)
M.I.T., Instrumentation Lab (2)
M.I.T., Air Force Field Rep (1)
M.I.T., Digital Computer Lab (3)
M.I.T., Lincoln Lab (2)
M.I.T., Servo Lab (12)
M.I.T., Statistical Services (1)

We never had further contact with the Patrick Air Force Base people, but some years later the Sputnik era and NASA TV coverage showed elaborate Mission Control facilities in full bloom. I like to think our earlier efforts at least had some influence on those crucial developments.

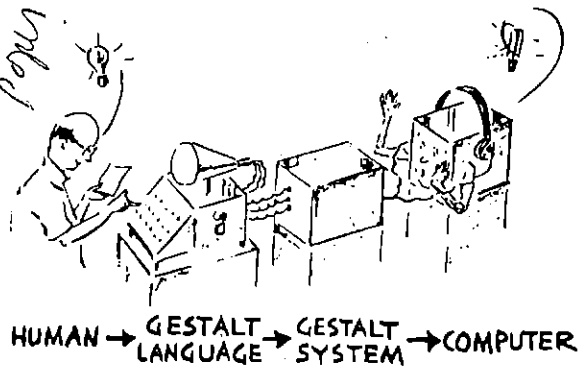
Section 4

GESTALT PROGRAMMING

The response to the two M.I.T. symposia and the clarification in my own mind that what we were building was indeed a new way to use computers prompted me to make this the topic of what would be my first professional paper. Even though all of Whirlwind may seem too huge to be a PWS, my view was and is that the PW aspect dominated the WS reality then available, so the theme of that paper and this paper is valid.

4.1 Early Views

My records don't indicate when I decided to submit a paper for the 1956 Western Joint Computer Conference (precursor to the Spring and Fall, and now the National Computer Conference series of what is now AFIPS, the American Federation of Information Processing Societies). I was Chairman of the Printing Committee for the November 1955 Eastern Joint Computer Conference, (EJCC 1955) held in Boston (we designed the conference logo, combining the logos of the IRE, AIEE, and ACM joint sponsors, which was used for several years thereafter) so I knew how the conferences ran. The earliest dated item I have (the date stamp is October 13, 1955 (Ross 1955d)) is a handwritten yellow-page draft reading "[The] Gestalt [System of] Programming -- A New Concept in Automatic Programming", with the [ ]ed words crossed out, along with an aborted opening sentence. Notice how that subtle and impulsive act makes the Title say that this is a paper about a generic concept of a type of programming. If the words were not crossed out, the paper could be about a systematic way to program -- a school of style -- or (even more specifically) the paper could be about my particular collection of programs constituting a system with that name. In the first two drafts I do use the name "Gestalt System I" in a couple of places. But when I see the cross-out now, I see it as evidence (matching the slant of the words of all drafts of the text) that I always had the generic concept in mind as the subject of the paper.



Communication from Human to Computer

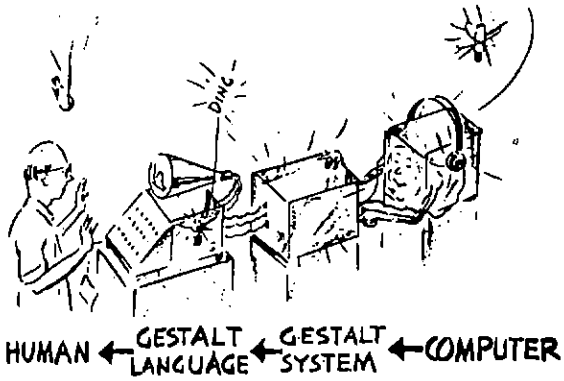
I remember that from the first rush of insight that so entranced me in that first exciting visit to Room 222 I knew that things would be profoundly different when man and machine were intimately coupled. I always had a broader view than just our data reduction application in mind. The earliest record I find on our use of MIV is in the Third Quarter 1954 Whirlwind report, which is suitably subdued, but also clearly is targeted to experimental programming:

"Work is progressing on another phase of this problem, using auxiliary in-out equipment both as an integral part of the data-reduction programs for monitoring purposes and as an aid to experimenting with improved computational techniques. Intervention registers and scopes will constitute the working parts of this system with efforts being made to provide the flexibility normally associated with analog computers. Present work is concerned with the development of routines to decode intervention-

register inputs and to utilize various specialized scope outputs. After merging these routines with the data-reduction programs, an elaborate logging scheme, using magnetic-tape output, will be written for permanent records of all operations and interventions during an experimental run." [WW3Q54p15].

But in an August 11, 1955 letter to my Oberlin classmate, Ben Scheff, urging him to come and work with me at M.I.T. following his Navy tour at the National Security Agency, more of my real feelings show:

"The work on fire-control system evaluation has been extremely interesting and has grown continually in scope and capabilities. Now we are really on the forefront of a whole new way of using large digital computers. We are in the fortunate position of being purely a research group and



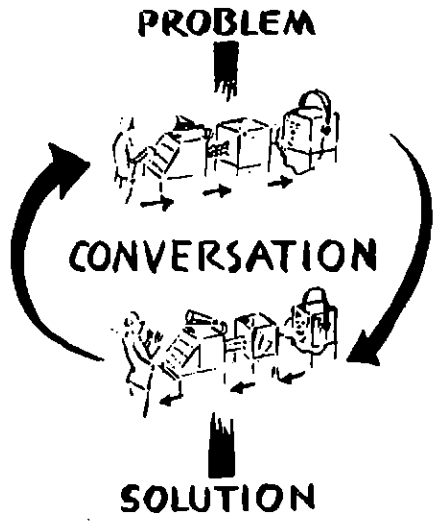
Communication from Computer to Human

never have to get our hands dirty with actual data reduction. Once we get a new technique developed and proved, we drop it or expand on it to new, more comprehensive work.

"A large part of our efforts go [sic] into developing new techniques and programs to achieve more flexibility in devision [sic] and testing new programs. The MDR report, which is enclosed, is one such program, and is a pretty good example of the kind of programming and philosophy we like to follow. The other special report on our Symposium gives some idea of the techniques we're now exploring. Both reports are now out of date but give the ideas.

"It is really amazing the complexity which arises in making the computer act in these ways, but will be a really nice system when it's done. We have an intermediate version running now which we demonstrated at the Symposium, but the newest version (almost complete) will run rings around it. The new system is really much more general than we now require but it will essentially allow us to program with programs while the computer is running instead of coding for weeks before a run.

"In a nutshell, what we're doing is fixing up Whirlwind so that we can manipulate its operation while it's running, turning on and off programs, examining and modifying so that we can use the computer as a research tool on an idea or concept level, rather than on the grungy coding level. Frankly, it's pretty exciting work ... I can honestly say that I don't know of any other work in computers and their use which is more interesting than what's going on here." [C55811]



Solution by Conversation

4.2 Initial Drafts

I have several distinct drafts of my attempts to write the Gestalt paper. The November 17 version actually submitted to Byron Bennett (SRI), WJCC Program Chairman [C551117] is completely different from the October 13 draft, and that paper, in turn, was completely reworked several more times before the final submission of our Project Technical Memorandum version (Ross 1956c) on February 1, 1956, for publication in the Proceedings. I think it is historically important to dwell briefly on the evolution of my expression of my then year-old, but still elusive ideas, because such an analysis will show my present PW versus WS theme in the struggles of its original context.

The original (October 13) draft jumps right in with the dictionary definition of Gestalt psychology: "The theory that ... events do not occur through the summation of separate elements, ... but through formed patterns of these, integrated units which function singly or in interrelation," and proceeds to say that standard programming practice is to be replaced

"by simple and unambiguous expression of the desired characteristics of behavior, the expression itself in effect tying together integrated units of computer behavior ..." (presaging today's declarative languages, functional programming, and the reusable components of software engineering). And still in the opening paragraph, "automatic coding systems are aimed at easy communication between the human and the computer, whereas a Gestalt system of automatic programming is aimed at easy conversation ..." [a phrasing with which I was particularly pleased, and which I retained in all further drafts]. (Ross 1955d) These are indeed the pertinent ideas, but the later drafts are much more roundabout in approach. At the time, even ordinary programming was still immature, and there was a complete lack of the common, shared culture we know today. The ideas were very hard to formulate and very hard to assimilate.

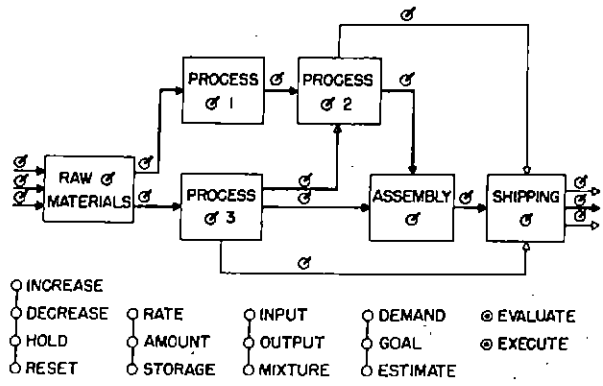
The second (November 17) draft (the basis for acceptance of the paper) begins with the following "Introduction:

"In any human endeavor there are three major phases: conception, expression, and execution. Gestalt programming is an attempt to make these three phases as nearly identical to each other as possible with respect to computer programming. In this paper the word Gestalt is used to mean a concept of a task to be performed by a computer. In a Gestalt system of programming the Gestalt, or idea, is expressed simply and unambiguously in a special language, rather than through the laborious assembling of machine-codes, pseudo-codes, subroutines, etc. Using a Gestalt system, the expression itself in effect ties together integrated units of computer behavior, which function singly or in interrelation, to achieve the desired effect. The purpose of a Gestalt System is to facilitate the transmission of general ideas as in a conversation, between a human and a computer, so that the maximum use of their respective capabilities can be made.

"After presenting the abstract theory of Gestalt Programming this paper discusses several Gestalt Systems in use today at the Massachusetts Institute of Technology and describes briefly the types of computer hardware which are best suited to this application." (Ross 1955f)

This was retained as the Synopsis for the final paper, but the body was reworked many times. The referenced MIT systems were the Whirlwind Comprehensive System (CS) itself (including mention of Director Tapes), the Whirlwind Marginal Checking facility (which required a technician to classify oscilloscope traces, etc., through MIV, and then isolated faulty tubes and components), and our own Data Reduction and Experimental Programming System.

In addition to various attempts to describe problems in which MIV was desirable (not an easy task in the hands-off, standard mode of submitting runs to a computer operator) I needed to have an example to illustrate the actual design and programming steps. My draft pages show one attempt using the LIP and RIP (Left Insertion Panel and Right Insertion Panel) and various buttons for CS computer operations, but that was not used. Another complete development says "Consider the launching of the man-made satellite schedule for the coming Geophysical Year," including the suggestion that a "part of the Gestalt Language might be a kind of joy-stick, so that the human could steer the rocket remotely." (Ross 1955e) But I decided that this example would seem too flashy, and instead worked out an example of Automatic Factory Control for the final paper (perhaps a prescient choice, since my soon-to-arrive role in APT was unknown to me then).



Gestalt Language for Automatic Factory

4.3 Oral Presentation (February 1956)

Then there follow endless yellow pages of my attempts to work out an oral presentation of the paper (Ross 1956a) ("The title of this paper has probably raised questions in your minds both about what I am going to say and why I have chosen to introduce a new word-usage into the already crowded computer terminology" -- scrubbed, fortunately!) I presented an interdepartmental colloquium version at MIT entitled "Talking with Whirlwind -- An Introduction to Gestalt Programming" on January 11, 1956 (rescheduled from December 14), [C56D16 to Bennett, and (Ross 1955f Addendum)] and finally managed to do a decent job as the opening paper at the San Francisco WJCC on February 7. Electronic Design magazine enthused:

"Programming and coding sessions at the Western Joint Computer Conference, San Francisco, February 7, 1956, were attended by overflow crowds. Highly significant papers were read. Here is a verbatim report of the impressions on [sic] the session chairman, Francis V. Wagner, group leader, engineering computing, North American Aviation, Inc.

"The paper on 'Gestalt Programming' by Ross, which opened the session, was a milestone of forward thinking. It refused to accept the common fallacy that human beings can contribute little when teamed directly with automatic

digital computers, and took the position that only the more versatile human mind can, at present, handle some parts of some problems. In the past, many people have noted that, to exploit such a partnership, there is required not merely a satisfactory language for communication from the human to the computer but a language which allows fluent conversation between the human and the computer. Mr. Ross went beyond merely stating this, [I thought it was original with me, at the time! -- D.T.R.] and presented evidence of some serious thinking towards analyzing how such a language ought to be designed. The principles which he stated and the illustrations which he gave of ways to design such a language, and of the hardware which might form the medium for the conversation should go far towards clarifying the problem for future implementation of these ideas ...

"Audience participation was distinguished more by quality than quantity. No more than three of four questions were asked of each speaker. [Wagner's overall impression of audience was that on the average they probably were not prepared to comprehend the concepts presented. -- E.D. Editor] Although these were put by highly competent people, and indicated careful evaluation of the paper, they were aimed more at underlining or amplifying certain points. In no case did they take violent issue with any of the statements or implications of the speaker, nor did they open up any controversial issues." (Electronic Design 1956)

Even the Scientific American expressed an interest in the paper. A suitable summary of my thinking at the time is the following quote from my January 12, 1956 reply to Dennis Flannagan, Editor, enclosing a copy of the paper. Actually I sent the final, TM version -- but I still was concerned about its adequacy in expressing my ideas. This probably was my own dissatisfaction with my MIT talk the day before, even though it was very well received. A few Lincoln people let me know by body-english that they thought I was plagiarizing other people's ideas, however. It was clear that they thought the ideas were important -- important enough to let me know their feelings -- but I, in turn, saw their upset as evidence that they really didn't see the differences between the specific Cape Cod incorporation of the man in the loop as an engineering necessity, any my generic Gestalt Programming idea. Somehow I wasn't expressing myself clearly enough, or they wouldn't feel that way, I thought. Hence I wrote to Flannagan:

"Since it is not stated too clearly in the present draft, let me try to summarize for you here, what Gestalt Programming is and where it fits into the established computer technology. As computer techniques have developed over the last few years, there has been a growing trend toward more sophisticated methods for connecting the human, who states the problem, to the computer, which is to solve the problem. Out of this trend has come, as a natural consequence of the maturing technology, a desire to use computers for solving problems which cannot be completely specified in terms which the computer can handle. These major trends are united into the general problem of using humans and computers together to solve problems. An analysis of this problem shows that more efficient and natural languages are required to bridge between human and computer, and that these languages must operate at the idea or concept level. In particular, a statement in the language must lead directly to the solution, in the same way that a Gestalt (in the psychological sense) unites at once, basic units into a single entity or pattern. The theory of Gestalt Programming is an attempt to set the initial outlines of the structure of these languages, and to indicate how to construct and use them. The physical content of Gestalt Programming is not new, but the emphasis and point of view is new, and is intended to clarify the directions in which these fascinating developments can go." [C560112]

But nothing further came from the Scientific American interest, which is too bad, because unbeknownst to me, at about the same time (and completely independently) George Price was in communication with the editors of Fortune magazine, preparing his landmark article on The Design Machine, which appeared in the November 1956 issue. Francis Bello, Science editor for Fortune wrote to me on March 27, 1957 that

"Price wrote us in November, 1955, suggesting his article and we had a rough draft in our office on January 15, 1956, which outlined the major features of his Design Machine. I mention this because it may surprise you that the article was so long in preparation before appearing in Fortune. (And certainly you would be entitled to wonder if Price had heard of your February paper.)

"What surprises me still more is that computer experts at I.B.M., Sperry Rand, and Lockheed, who read Price's paper for us, and checked his computations, had not heard of -- or at least gave no indication that they had heard of -- your work in this area.

"From the beginning our concern was that Price's ideas were too blue skyish for serious presentation. The world keeps moving faster than Fortune can keep up." [C57327]

To which the only comment (valid today, as it was then) is -- it does indeed. Before I address Price's article, there are more items from our own work to be covered.

Section 5

CONSOLE DEVELOPMENTS

With the PW component established, next comes the WS component of my PWS theme. I can cover here only the earliest developments of what was to be a long term evolution of computer graphics and keyboard language techniques, both hardware and software, well into the 1960s, at MIT.

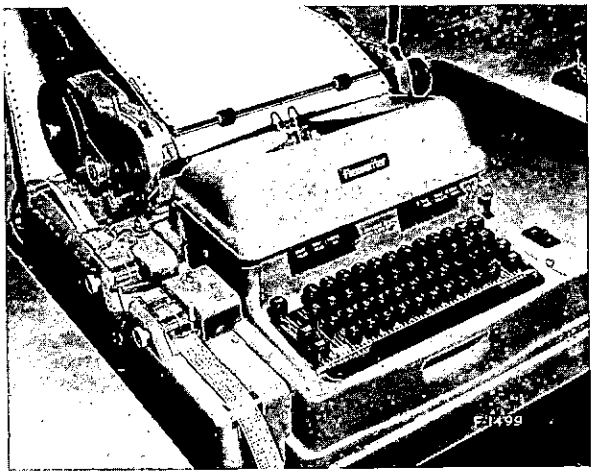
5.1 Direct Keyboard Inputs

It's funny how memory plays tricks on us. For years I have known that it was at my instigation that direct keyboard input to Whirlwind was installed (beginning the chain of events that eventually led to the Compatible Time Sharing System (CTSS) and Project MAC at MIT in the 1960s). But at the same time, I thought that all of our MIV and Charactron console design and installation for the Eglin Field 1103 had come after and had been based upon what we already had going on Whirlwind. The records show that I was mistaken. The Whirlwind keyboard proposal actually followed my proposal for the 1103 keyboard. That came as a surprise to me, when writing this paper.

The earliest date I find in my own files regarding the ERA 1103 is a collection of general information (brochure description and Installation Instructions -- where to install the motor generators and air conditioning, etc.) bearing date stamps of John Ward and myself of November 8, 1954. (ERA 1954a and b) Two yellow pages of notes about the order code indicate that this was my first introduction to what was in store. The earliest Charactron information is a collection of Convair Charactron Project documents on the Model 708 Charactron Display Console, the Model 100 Computer Readout System, and the Rapromatic Camera and projection system, all with my date stamp of October 18, 1955. (Charactron Project 1955)

By December 14, 1955 I had written a memo on "Proposed Equipment Modifications for the Pre-B-58 Data Reduction Programs". (Ross 1955g) It recommends that AFAC should procure an additional 4096 registers of magnetic core memory beyond the 1024 electrostatic storage registers that were standard for the 1103, and "that the photoelectric tape reader and the Flexowriter connections be modified", in addition to installation of the "Type 70-A Charactron with photographic facilities under computer control, reported already on order", as well as "the Intervention Switch facilities, reported already on order."

Engineering Research Associates (ERA) definitely was a company of (outstanding) engineers -- not programmers. I believe the AFAC 1103 was one of two "Serial One" first models of the ERA 1103, (Cohen 1985) and I encountered several features that were good engineering ideas, executed in such a fashion as to make programming and use most awkward. The 1103 had a 36 bit word length, and an "Arithmetic Section" with Accumulator (A of 72 bits), Shifting Register (Q of 36 bits), Exchange Register (X of 36 bits), and two Input-Output Registers (IOA of 8 bits and IOB of 36 bits). The binary form of 7-channel punched paper tape, called "bi-octal", was prepared off-line using a box full of relays with a 10 character per second mechanical tape reader on one side and a mechanical tape punch on the other. A number of empty sockets into which jumper-wired plugs could be inserted allowed the



Flexowriter with 10cps reader (front) and punch (back). Keyboard was not used for Whirlwind direct input until summer of 1956, which led to later timesharing projects at MIT.

user to control the tape transformation between the two. Absolute address assembly language tapes punched on an off-line Flexowriter could be converted to bi-octal with one set of plugs.

We contracted with the MIT Digital Computer Lab to have a WWI-1103 Input Translation program, based on CS, built so that we could write 1103 programs with "flads" (floating addresses) etc., translate them on Whirlwind, fly them to Florida, and debug and use them there. [WW1Q55p56and79-81] The 1103 itself had no such software. I wrote a companion 1103 program, handling only absolute addresses, to support our on-site debugging efforts. (Ross 1955b)

For this early machine, things were pretty awkward, the ERA photoelectric tape reader (PETR) could only be started manually, the PETR was connected directly to the Q register, thus entangling the Arithmetic Section unnecessarily, and worst of all, as my memo says:

"The bi-octal tape format now in use does not allow the use of programmed checks to insure that the information was read from the tape correctly. Since the read-in program for reading bi-octal tapes is now wired into the computer, the increased flexibility required to change tape format is not presently available.

"There are two major approaches to the problem of placing the photoelectric reader under computer control. A Ferranti tape reader could be purchased and installed exactly as in other 1103 computers. This is the best, but most expensive solution. The other possibility is to put the drive motor of the present ERA tape reader under computer control, and to have the photocells read into IOA (or IOB) rather than into the Q register. The only disadvantage to this inexpensive method is that a few inches

of extra tape feed-out must be placed in the tape wherever it is to be stopped, since the ERA reader is not a fast start-stop device like the Ferranti reader. However, this method was used successfully with a similar ERA reader for several years on the Whirlwind computer at MIT, before the Ferranti readers were obtained." [p. 2] (The inexpensive method was chosen, and the WWI-1103 Input Translation program was forced to ensure the proper blank tape sections were present.)

The memo then continues with similar observations regarding the keyboard:

"6. Flexowriter Input

"To augment the Manual Intervention facilities mentioned above, it is suggested that the Flexowriter presently attached to the AFAC 1103 computer be connected to act as both a keyboard input and mechanical tape reader input device. This method is considered preferable to the purchase of special keyboard input devices at this time because, again, it should be very inexpensive, and will give considerably more flexibility than any commercially available keyboard of another type.

"An essential part of Manual Intervention Programming is that decimal (or octal, or English) information can be communicated directly to the computer. In use, this system would use intervention switch settings to tell the program how to interpret the Flexowriter keyboard input at any given time.

"The mechanical tape reader probably could use the same circuitry as the keyboard and would be used to read control tapes (while data or program tapes are in the photoelectric tape reader) so that manual control will not be required for routine operation of the computer system, which will result in greater reliability.

"A comment contained in 1103 Newflash No. 15, December 7, 1955, seems to imply that the keyboard of the Flexowriter on the Ramo-Woolridge 1103 is already connected in this way, so that the required engineering may be already available. The comment does seem to imply, however, that the 'directly connected typewriter' uses the Q register. It is much better to use IOA (or IOB) for this purpose so that normal operation of the arithmetic element is not impaired. It is very possible that both the keyboard and mechanical reader could share a major portion, at least, of the circuitry connecting the photoelectric tape reader to IOA (or IOB), with resulting savings in engineering and hardware." [p. 3]

It is interesting to me that even though I chide the engineers for their lack of vision, I suffered similar myopia regarding how to establish controls. I'm sure that it would not have taken significantly more programming to have an opening character or so to specify what type of information was on the tape, instead of using MIV switch settings. This had, in fact, already been done for the logging of CS tapes on Whirlwind, after all, when Director Tapes were introduced. Our "modern" views take time to evolve and develop, I guess.

In any case, this December 1955 memo shows that the AFAC 1103 keyboard suggestion formally came before the Whirlwind keyboard suggestion, and in a fashion that surprised me a second time, in my researches for this paper. I had found in my files my memo 7138-M-144 "Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer", dated March 1, 1956, (Ross 1956c) and my memo 7138-M-148 "Flexowriter Keyboard Input to WWI, Preliminary Specification", dated March 14, 1956. (Ross 1956d) But digging still deeper I found that my memo 7138-M-160 "A Proposed Flexowriter Input for the Whirlwind I Computer", dated April 30, 1956, (Ross 1956f) was the formal proposal finally sent on to Lincoln Labs for formal consideration, with the concurrence of the Scientific and Engineering Computations Group and "several groups in Lincoln". So at this point in my sleuthing, I was convinced that the actual sequence for my on-line keyboard suggestion definitely was first 1103, then Whirlwind.

The second surprise came from a detailed reading of collected draft papers for a May 8, 1956 revision (to "144A") of my 7138-M-144 March 1 memo, this time done by John Ward, (Ward and Ross 1956) for formal forwarding to AFAC and our WADC sponsors. In that revision, which includes John's redesigns of Whirlwind drawings for Intervention, Activate, and Audible Alarm circuits, is a large fold-out drawing entitled "Keyboard Read In System, WWI", showing Lou Norcott, "Eng." and Larry Holmes, "Appd.", both dated "2-27-56" -- with the hand-scribbled notation "see him [Norcott] or Doug R. O.K.". The interesting surprise to me is that this shows that I already had been active during February, and probably before, working behind the scenes with the Whirlwind engineers to confirm feasibility and costs before sticking my neck out with formal suggestions. Therefore the actual fact is that I can't sort out which came first, my thoughts regarding the 1103 or Whirlwind. My guess is that they surfaced almost together. I.e. I knew general keyboard input was needed for Gestalt programming, but the fact that the ERA 1103 already had an on-line keyboard of some sort triggered me into action. I first worked the idea out on the basis of Whirlwind, working with Larry Holmes, but the first actual writing was about the 1103.

Larry and I played the game to the hilt, for I also have a short interoffice memo from him to me, dated April 3, 1956 (Holmes 1956) which begins:

"I received a copy of your Memorandum 7138-M-148. The attached chart illustrates the five different plans that you might consider in the determination of a decision as to which installation you will request." An then goes on to conclude:
"I sense that it would probably be the middle of June before the chosen facility would be operational.

"If any additional information is required for your subsequent memorandum to management, I will be pleased to procure same."

Bobbi: place this remin history w/ I/O devices & SW interaction

5.2 Microprogramming for Whirlwind

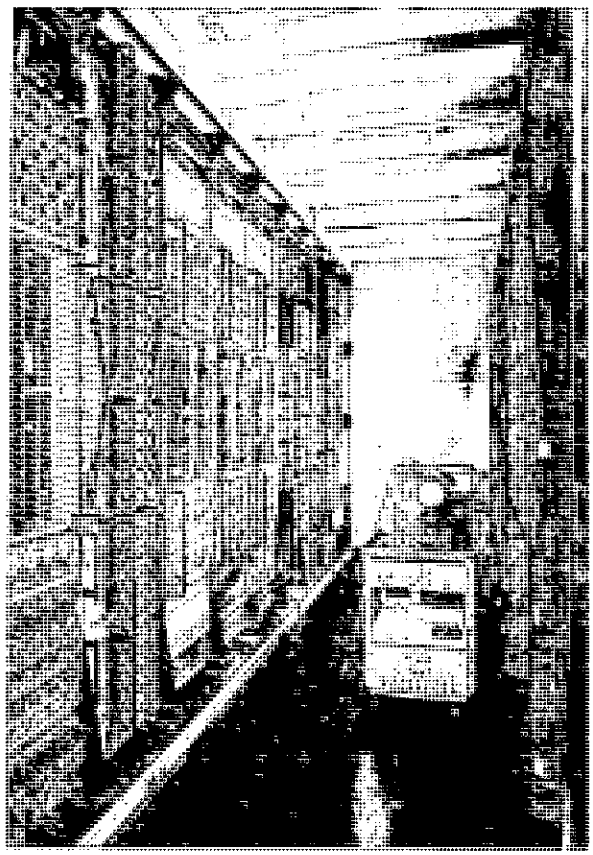
It was not all that unusual that I was in close touch with the Whirlwind engineers, for another one of my on-the-side, recreational projects at that time concerned a plan I had to modify the internal logic of Whirlwind I to allow experimentation with microprogramming. This idea had come to me in the same spurt of inspiration when I studied the Lincoln-only, si-based input-output lines that led to the original Director Tape proposal. I discovered that not only was the mechanical tape reader still available (as I have already covered), but much more importantly that there was a whole additional flip-flop register, called PR -- the program register -- just sitting there, completely unused! The PR had temporarily held the memory address for Electrostatic Storage operating and was no longer required for magnetic core memory. It still had all the wiring that connected it to the WWI Bus, and had three outgates per digit! What a treasure trove! In those days of discrete components and tubes, flip-flops and gates were very expensive (several hundred dollars per bit), and here was all that gear, cut out of use by the modifications to the main machine, but all ready to be fixed up if I could find a use for it.

Of the papers I still have, the earliest date stamp I find is that of "Engineering File #2, August 18, 1955" on copies of large drawings of "Timing Diagrams, WWI", Digital Computer Lab drawing E-51744, and "Control Pulse Output Connections, WWI", DCL drawing D-5317? (torn off), which were made for my use. My own vellum master drawing of "Proposed μ-Operation Schematic for mi Instruction, WWI" is dated April 5, 1956, and numerous copies show my steps of refining my design after that date. (Ross 1956e)

On March 1 and 2, 1956 a seminar on Microprogramming, sponsored by the MIT Digital Computer Laboratory was held with papers listed by H.R.J. Grosch (GE), D.J. Wheeler (Cambridge U), N.R. Scott (U of Mich.), L.C. Hubbard (IBM), B.D. Smith (ERA), J.J. Eachus (Datamatic), H.P. Donahue (NSA), G.M. Hopper (Remington Rand), J.W. Wegstein (NBS), and J.W. Carr (U of Mich.). (MIT Dig. Comp. Lab. 1956) I was not even on the list, because the Digital Computer Lab organizers didn't know about the hobby project I'd been doing only by myself, with a few interactions with the Lincoln Lab Whirlwind engineers. I hadn't even gotten an invitation to the seminar, but when I heard about it, I was welcomed, and went.

With a June 6, 1956 date stamp, I got pages of a typed transcript of the seminar from Frank Helwig which shows that at the last minute I had been squeezed into the program before Wheeler's second talk. As is usual for my oral speaking, the transcript is almost incomprehensible, but so is the extended ensuing discussion by others.

I did manage to complete the design of the proposed mi microprogramming instruction and



Control Matrix of Whirlwind. Wires, diodes, and gate boards laid out translation of 5-bit instruction code into 120 Command Pulse Output (CPO) micro operations.

my papers do include a complete outline of the report I intended to write, but never completed. (A "to-do" list, buried in these papers starts with "Flexo keyboard memo - Fri; AFAC int sw. diagram; Pre-B-58 MIT meeting; Gestalt Paper to AFAC, etc.; mi-WWI memo" and a slew of other items, including "Numerical Control Personnel", so I'd guess the date was around May, 1956 when the outline was done.

The idea of the proposal was as follows: When you walked inside the bowels of Whirlwind, down one aisle was a mass of bare copper wires and boards of components, neatly arranged in a matrix. First came five bare vertical wires, and then 32 long, horizontal, bare wires going down the length of the aisle. These, in turn, overlaid 120 vertical bare copper wires, each of which terminated incircuitry boards arranged along the bottom, by the floor. There also were eight coaxial cables running horizontally, with one or another feeding each of the lower circuitry boards. These cables carried one of eight time pulses, and the original five vertical wires supplied the 5-bit instruction code of the instruction now being executed. This was the Control Matrix of Whirlwind. The neat thing was that soldered right there in front of you, at the various crossing points of the five vertical wires and the 32 horizontal

wires, was a collection of germanium crystal diodes that spelled out the binary number system, so that each of the horizontal bare wires represented one of the 32 instructions of Whirlwind. To understand what each instruction did, you had merely to similarly read the pattern of diodes soldered between its horizontal wire and the 120 vertical wires. The 120 vertical wires each selected one of the 120 CPO Units (Command Pulse Output gates) of Whirlwind, each of which could do some micro operation (such as clear the I/O Register, read from memory into the A Register, etc.) when its corresponding time pulse arrived. In a nutshell, my plan was to use the flip-flops and gate connections of the PR to dynamically compose instruction steps from a selection of the existing CPOs -- as though a new horizontal line had variable diode connections to the CPOs.

The report outline indicates that I found a subset of 39 CPO micro operations that would be useful in various combinations. My design allowed any number of them to be chained together on time pulses 6, 7, 8, and 1, interspersed with memory addresses and I/O operations. Thus microcoded instructions of any length and complexity could be composed and executed, interspersed with standard Whirlwind instructions. As the outline says "Very fertile field for problems which can use micro-program in MIV Gestalt Programming." The report was never written, however, and the change to Whirlwind was never actually proposed (I had in mind a big gang switch that would include or exclude the mi instruction in Whirlwind's repertoire) because when I programmed various test cases with microprogramming and then with ordinary Whirlwind programming, there was very little to be gained. The trouble was that Whirlwind was so closely designed (to save expensive gates, etc.) that the A Register formed an insurmountable bottleneck. As the outline says ("md" is "multiply digits" -- a masking operation):

"Fundamental difficulty with present WW construction for this type of application since only entry to AC is thru AR so that usually AR is not indep. of AC. Also this necessitates use of more steps than really required since first must load AR before can work on AC. Also would be good to have a working operation built into circuitry to give effect of md's since in logical ops this is often fundamental but is lengthy μ-op-wise or md-wise since another mi is necessary to get back to μ-mode.

"Conclusion:

"Study of this report and independent investigation of this kind of approach to μ-prog is very instructive on problems involved. If more major changes in WW structure were acceptable might be able to obtain a useful gadget. Best bet, however, is to contrive basic idea of investigating μ-computers by similation without changing WW."

So my grand scheme for microprogrammed MIV and Gestalt Programming came to naught.

5.3 Keyboard Input for Whirlwind

The direct Flexowriter input for Whirlwind was built, however, and could be operated from either the E31 Console or from Test Control, as I had suggested. The Second Quarter 1956 Whirlwind Progress Reports says:

"Since 1954, the MIT Servomechanisms Laboratory has been using the WWI manual intervention and display equipment in the development of high-speed data reduction techniques. In order for them to expand their research into computer applications, it was essential that more versatile manual inputs be made available on the WWI computer. Besides requiring additional on-off switches, many of the new programs will be so complex and will require so many parameters that the only reliable way to instruct them will be to use specially designed mnemonic languages and translation programs. In order to have this general language structure available on a manual intervention basis, it is necessary to have a keyboard such as a Flexowriter for direct input to the computer.

"The MIT Scientific and Engineering Computations Group have contemplated the following applications for the new facility:

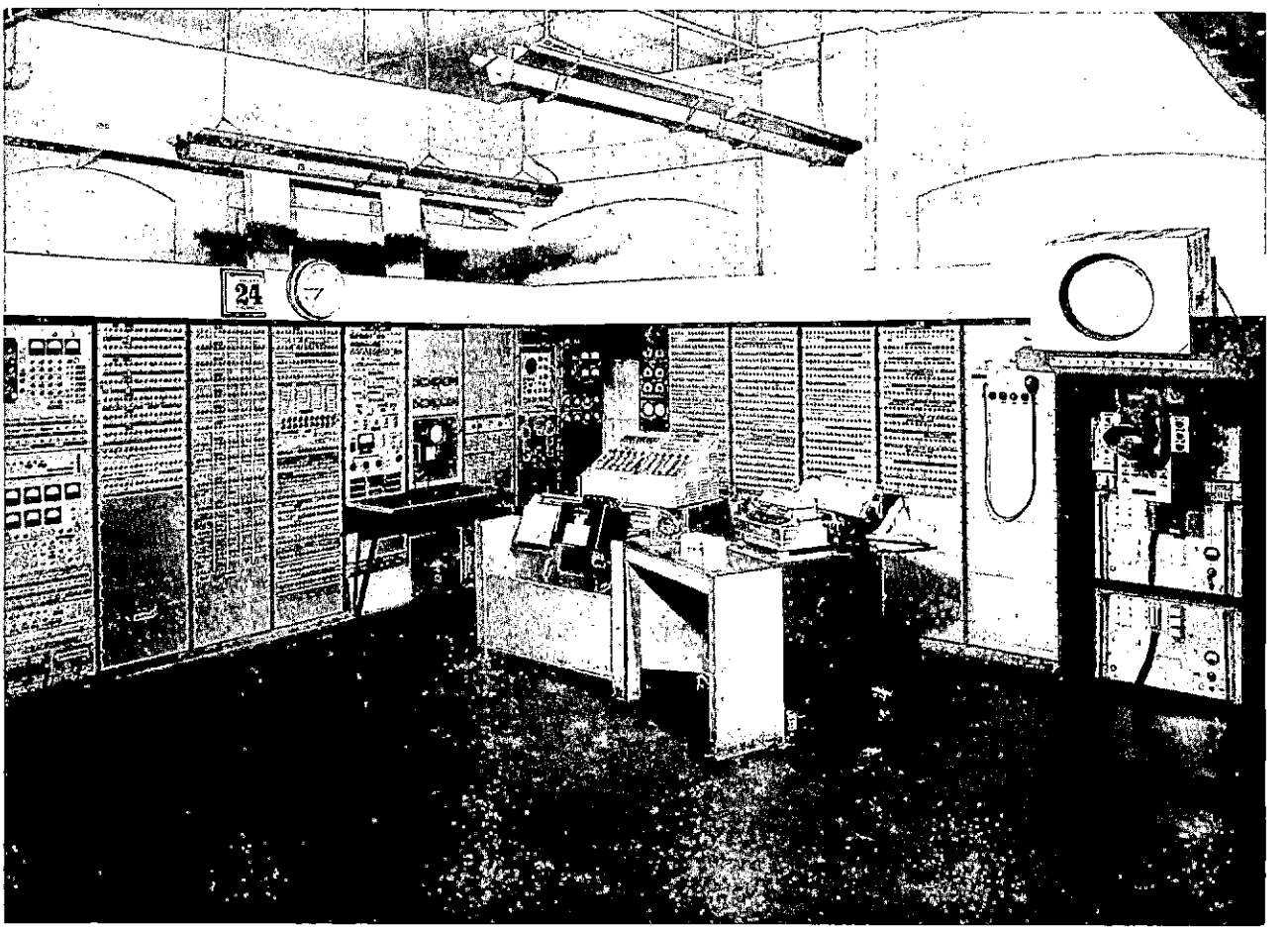
"1. Demonstration programs would be a great deal more effective if this form of input were available for control purposes.

"2. Typewriter input for Comprehensive System Flexowriter and post-mortem request tapes. Short program modifications and post-mortem requests can presently be inserted in the insertion registers. However, errors are easily made because the required vocabulary is awkward. A typewriter input facility would make available a normal mnemonic vocabulary for such purposes.

"3. Experimental use of a typewriter facility for direct operator control of the computer. Here we would consider using the typewriter to replace the button-pushings required of the operator during normal operations. Vocabulary similar to that of director tapes and performance requests would be devised for these purposes. This could easily prove to be an extremely convenient and efficient method of computer operation.

"The new input installation will be available for use by 4 July 1956. Much of the information to be inserted via the keyboard will be the same as is now introduced via a free running photoelectric tape reader using punched paper tapes. The keyboard input will also be treated as a free running device, i.e., selection of the facility by the computer may be followed by an arbitrary number of read instructions, each of which reads the next character which has been struck on the keyboard. The total equipment requirements amount to 15 relays and 20 tubes." [WW2Q56p63]
My original M-148 proposal specified that

"Since much of the information to be inserted via keyboard will be the same as is now on paper tape, it would be best to have the keyboard operate in a free running mode, i.e. an si selecting the keyboard may be followed by an



Whirlwind Test Control Console (1/2/57) showing MIV panels for computer operation behind Photoelectric tape reader (PETR) and Flexowriter. Monitor (above) and camera (below) output scopes are on right.

arbitrary number of rd's, each of which reads the next character which has been struck on the keyboard. In this way, programs such as the entire CS system can be modified to accept keyboard rather than tape input merely by changing one or two si instructions." (Ross 1956d, p. 2]

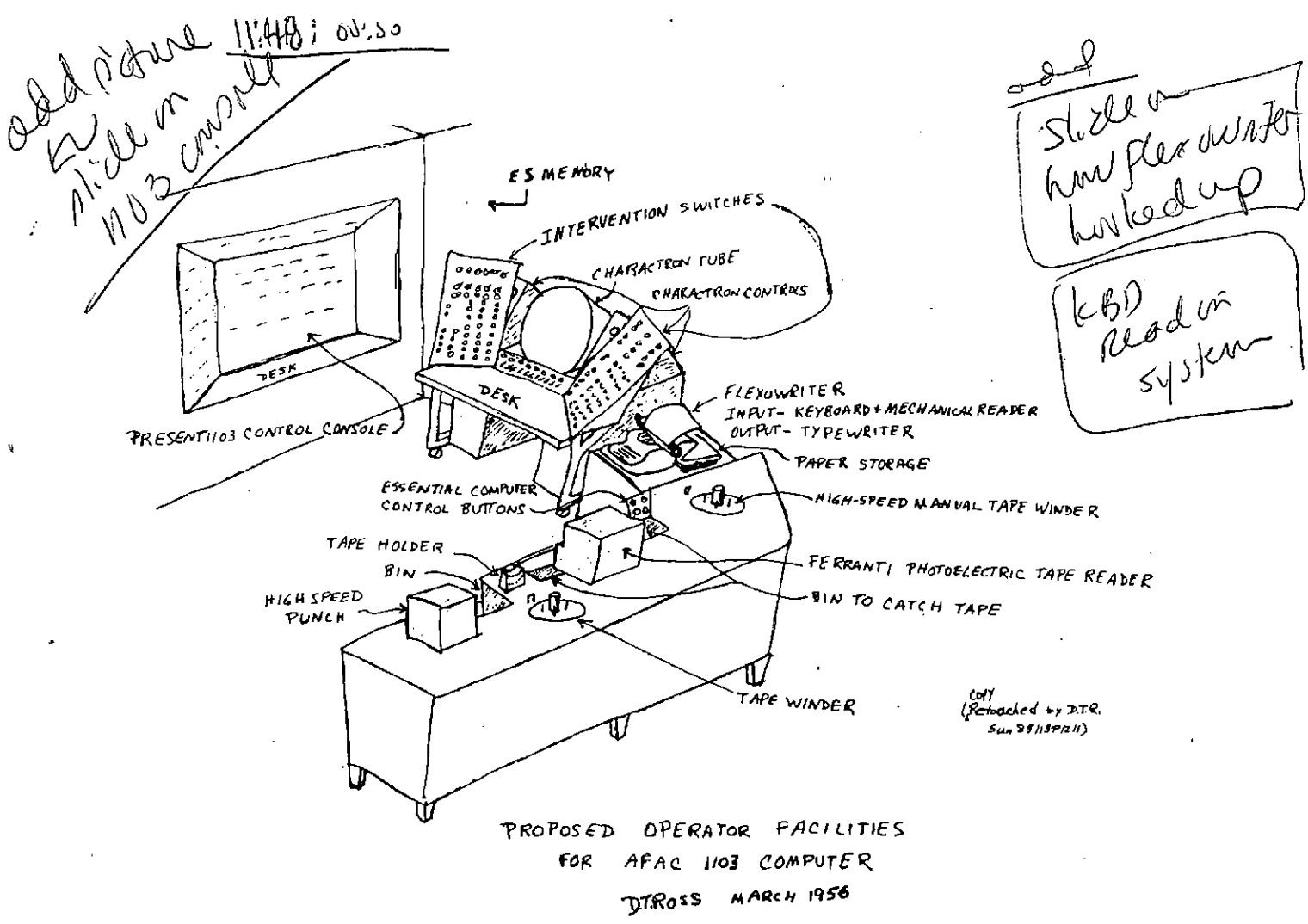
The existing Test Console Flexowriter was selected by one si, and another Flexowriter on a wheeled table plugged into the Room 222 E31 Console and was selected by a different si. We used it some with our SLURP developments, but Lincoln people didn't take advantage of it, to my knowledge.

While digging through my papers, much to my surprise I ran across a Division 2 Lincoln Lab memorandum dated February 2, 1959 (three years later) on "The Direct Flexowriter Keyboard Input System at Whirlwind I (Barta)" by C.F. Brackett in which I discovered that

"A pushbutton on the direct output Flexo table [in Test Control? - D.T.R.] allows the Flexo keyboard to be used as a direct input device. If this pushbutton is held depressed and the Flexo keyboard operated, the Flexo code for that character will be set up in the keyboard Input Relay Register where it will be available for later transfer to the In Out Register ... Since the keyboard Relay Register is cleared of old information during the read in of new information, the operator should normally wait until the indicators are extinguished before typing the next character." I never knew it had such a button. Maybe it was only on the Test Console Flexowriter. If the Lincoln memo indicates what actually happened when my idea was passed through the Lincoln approval cycle (I never had occasion to use it myself, as I remember), no wonder that first installation was little used. The button would force pick and peck typing! This equipment did, however, serve as the basis for later direct keyboard inputs to the IBM 704, 709, and then 7090 computers at MIT, which led to the initiation of time sharing.

5.4 The Charactron/MIV Console for the AFAC 1103

The Charactron/MIV console for the Eglin Field AFAC 1103 as finally proposed had two 36 bit Intervention Registers, one 8 bit Activate



Sketch for Charactron/MIV Console for ERA 1103 at AFAC, Eglin Air Force Base. Built by Stromberg Carlson based on Whirlwind experiences of Servo Lab Data Reduction Project.

Register, one 36 bit Indicator Light Register, a 7 bit MIV Flexo Register, and two Alarm Buzzers. (Ward and Ross 1956) John Ward and I supplied reworked Whirlwind drawings for all these items plus the Whirlwind drawings for the Ferranti PETR and manual tape winder at various times. All of these were forwarded to Stromberg Carlson, San Diego who reworked the designs further and incorporated them with a 19" Charactron for viewing and a 7" Charactron with camera, in the final, delivered installation.

My daily resumes began at about this point, and various phone calls, letters, and visits to MIT by Ed Allmon and various people from Stromberg Carlson are recorded from October 31, 1956 on. Topics included the vector generator and tape winder [R561031]; Ferranti PETR [R56116]; PETR again, plus the fact that they got the contract from AFAC [R561115]; visit regarding Flexowriter, tape reader, and 1103 MIV system [R561126-27]; connectors and pushbuttons [R561210,-11,-12]; calls from their psychologist regarding switch types and layouts (considering a black background for the lights, etc.) [R561218-19]. They finally sent a design study document to us on January 1, 1957, (which I don't have), but still there were more Flexowriter questions a week later [R570118]; the Ferranti automatic turn-off to save motor wear [R57314]; a press release [R57417]; request for a San Diego training program for use of the MIV equipment [R57429]; and finally "the Charactron and MIV equipment turned up at AFAC today" -- on August 14, 1957 [R57814]. But I have no photos or descriptions of what finally was delivered.

A letter from John Ward to John Moser (Strombreg Carlson) on October 24, 1957 congratulates him "for a very fine job" and says we "hope to be making good use of it within a month's time." [C571024]

John Walsh (one of my programmers, and still with SofTech) labored many months on an elaborate 1103 program for extendable, labelled plots of functions for the Charactron as a major part of the AFAC system. (Ross and Ward 1957) But AFAC programmers always were too short of time to do much elaborate MIV programming. The system was indeed used for the Pre-B-58 and B-58 tests, but I don't know any more about it than that. It's claim to fame really is that it was the first working work station explicitly designed for general purpose use.

5.5 MIT MIV for IBM Computers

In the Summer of 1956 preparations were well under way for the installation of the IBM 704 computer in the M.I.T. Computation Center. This also was the time that my Computer Applications Group was officially christened, as we acquired responsibility for the (then unnamed) APT Project in addition to our other projects. My first and second memorandam for the Group (Ross 1956g and i) concerned an assessment of the impact of the 704 on our work. The second memo begins:

"This memorandum is a supplement to Memorandum CA-M-1, 'Servo mechanisms Laboratory Requirements for Computing Facilities 1956-1957'. That memorandum stated that Project DSR 6873, on the development of automatic programming techniques for numerically controlled machine tools, would be transferred to the IBM 704 computer as soon as it became available, but that the work of Project DSR 7138, on the application of digital techniques to airborne weapon systems, would continue to need the facilities of the Whirlwind Computer System because of its dependence on manual intervention techniques. This memorandum considers in more detail the reasons why it is felt that the Whirlwind I Computer must continue to be used, by describing the system of Whirlwind programs which are now being developed for this work, and then considering the modifications to the proposed IBM 704 system which would be required to make this type of system possible using that computer." [p. 1]

Then, after a description of SLURP, it goes on:

"The IBM 704 Computer

"Aside from the large amount of programming which has already been completed, it is apparent that SLURP is independent of the particular computer used, provided it has the appropriate input-output devices. The manual intervention features of the system are central to the philosophy of the system, since without them it becomes merely another elaborate computer programming system. The true significance of the system is that it allows a programmer to conceive of a new method of solution and maintain his momentum and initiative on the problem, unencumbered by the restrictions of the coding system of the computer. In other words, the programmer is permitted to program with programs, and 'program with ideas'.

"It would be extremely desirable to have the larger storage capacity and longer word length of the 704 computer for use in SLURP. It already appears that the capacities of the Whirlwind Computer are being taxed by the size and complexity of SLURP. The present incomplete version contains approximately 10,000 instructions in addition to the 20,000 or so in the comprehensive system, and to this figure must be added several thousand registers required for the storage of a satisfactory amount of data for the problem. Since a large portion of the programming for SLURP consists of generating and interpreting coded information, the short word length of the Whirlwind Computer becomes awkward on occasion, and slows down the operating speed of the system due to increased complexities. Since the Share assembly program and the algebraic coding system being developed for the 704 installation by Computation Center personnel [never carried out] will be more modern and flexible than [p. 4] the Comprehensive System in some respects, and may be expected to expand to include those features of the Comprehensive System which are unique and desirable, it is felt that SLURP efficiency could be greatly improved by the use of those systems. Except for the fact that the IBM 704 will not have manual intervention and oscilloscope-type input-output devices, it would be very desirable, and probably worth the additional programming effort, to transfer SLURP from the Whirlwind Computer to the 704 Computer.

"There appears to be some indication that there is some interest in applying this type of equipment to the 704. The Servomechanisms Laboratory should be active in support of this thinking. Systems such as SLURP seem to be the next logical step in development of improved programming systems for modern computers. The fact that work for Project DSR 6873 for the automatic programming for numerically controlled machine tools is being formulated and solved within the SLURP structure demonstrates that the philosophy of problem formulation and programming which is embodies [sic] in SLURP is by no means restricted to the type of military problems which have fostered its development and for which it is primarily at present intended. The possibility of detailed research into automatic process control and managerial business decisions should provide ample justification for the active consideration of these techniques as an integral part of the MIT Computation Center facility.

"Conclusions

"It is felt that the problems presently being considered in the Servomechanisms Laboratory cannot be successfully solved without the use of SLURP so that until an equal facility becomes available elsewhere this work is committed to the Whirlwind Computer. It is felt also, however, that the concept of systems such as SLURP is a significant advance in the use of computers as data processing devices, and that active consideration should be given to this type of system as an addition to the facilities of the MIT Computation Center in the near future. It would be necessary to expand the equipment planned for that facility to include manual intervention input devices and oscilloscope-type output devices in order to realize this type of system. The experience gained from the use of this type of equipment in this application in the Servo Lab SLURP system, as well as the experience of the Sage system development, should provide an adequate foundation for the planning of a Computation Center facility of this type. The potentialities of this type of a system as a research tool for experimentation in the newly unfolding area of

general data processing are very great. It is felt that it would be in the interests of the Institute as a whole to pursue this line of thought." [p. 5]

My resumes show that on November 1, 1956, F.J. Corbato (still a professor at MIT) "called to inquire about Servo Lab people giving talks in Digital Computer Center [sic] seminars. He is now on their staff and is in charge of these seminars. I said that we might talk about SLURP and our general problem solving system." [R56111] On November 6 "should possibly consider obtaining support from Mission Director Project [a new component of John Ward's Bomber Defense project] for attaching MIV equipment to the 704 ... Will IBM let such equipment be attached to the computer and also how do we get enough computer time to make the investment useable?" [R56116] This was a problem we later solved by joining as a co-sponsor of the MIT Cooperative Computer Lab's IBM 709 in Building 10 in the early 1960s.

On November 27 I gave my Comp Center talk on "SLURP: An Experimental Human-Computer System for Problem Analysis and Solution" [R561114], and the interesting note is that Wes Clark (then doing TX-0 at Lincoln) suggested, in the discussion, "that the switches be programmed on the tube and thrown by means of a light pencil [sic]. In this way a programmer can do all of his own human engineering of switch layouts as well. [We later called this "light buttons" (Ross and Ward 1961, p. 80).] Slurp is already set up so it could do this operation merely by writing an appropriate Slurp program. Wes Clark would like to have several copies of the Gestalt paper and I will plan to take them out when we visit TX-0 tomorrow." Also at the talk, Frank Verzuh (Oirector of the MIT Comp Center) "said he was interested in talking with me about getting this kind of facility for the Computation Center" [R561126-27]. And the next day "Gave a stack of Gestalt papers to Wes Clark. He seems to be quite interested in what we are doing and there may be a possibility of pooling our interests if not our efforts in making TX-2 specifically designed for Slurp type of problem solving. Should plan to discuss this with him and his group in the near future." [R561128] We did collaborate on light pen design and improvement, under John Ward, after TX-0 was moved to MIT (I heard it was scheduled to move on June 27, 1957 [R57627]), but actual collaboration never took place.

Not much happened during 1957 on these ideas. TX-0 moved from Lincoln to ESL, with Earl Pugh in charge. In September, Whirlwind "passed into Lincoln's hands" [R5794-9] and we continued our developments there and on the AFAC 1103, while doing APT work also on the 704. By year end "A summary-progress report for the mission director is scheduled ... [Writing about SLURP will serve to] explain the whole problem-solving philosophy which I have been evolving over the last several years. [This will serve] as powerful ammunition for battling to keep Whirlwind going for a long time." [R57124-58019]. I cover the Gestalt Programming to SLURP transition in the next section, briefly.

In January 1958, Peter Elias (then EE Department Head) suggested to John Ward that I should get together with John McCarthy, Marvin Minsky, and Richard Marcus (all then at MIT's Research Lab for Electronics, RLE) about my use of MIV Techniques, [R58019-28] and John Ward wrote me a memo to that effect. (Ward 1958) John's memo says Dick "is working actively on function display under Manual Intervention Control." Later Dick came to work in my group and still is at MIT. By January 30, I finally arranged to have a lunch with McCarthy, which then transferred to Minsky's home in mid-afternoon, "and we talked some more until around 2:30 in the morning. Oliver Selfridge's group at Lincoln with whom Minsky works has $25,000 a month which must be spent on equipment, not manpower ... They are presently thinking of getting the [standard IBM] real time package for the Lincoln 704 and are interested in investigating whether the Convair MIV package that we worked out for AFAC would be within their budget." [R580131] I don't know whether Oliver contacted Convair directly or what he ended up doing, but this session did begin a long, sporadic interaction on equipment matters with John and Marvin. A week after our long session I loaned John McCarthy "our information on the Stromberg Carlson [Convair] MIV equipment ... informed him of our present investigations and said .. perhaps he would be a good one to work on the automatic programming features since he has been working quite a lot on a super compiler with all sorts of logical statements, possible in it which may fit in well with the system we have been working on. We shall see." [R5827] LISP was in the wind, and I was in the midst of APT, of course.

Again nothing happened regarding MIV equipment for months, although a talk I gave (on APT) at IBM Kingston Labs evoked brief interest on their part in scope input and MIV for their IBM 704. [R58815-20] In September, Dick Bennett (who then had a consultant company) was under contract to Lincoln to upgrade the Whirlwind Utility System, and he thought Frank Heart's group "also should be interested in being able to operate everything from the Room 222 console ... We also discussed ... putting things under direct typewriter control." [R58924] But most of Dick's plans were not carried out.

Even though the preparation for the APT press conference were pressing, I was concerned about the facilities for follow-on work, so in early January 1959 I said

"Whirlwind definitely is going to close on June 30 and since it is poor politics to fight it at this time, we are going to go along with it, so that we have to find a substitute. My thought is that by combining TX-0 and 704 we could have a really fancy facility since TX-0 with its newly expanded memory, which is not in yet, of 8,000 words should be able to do [display and light pen support.] ... We are having a meeting .. between [Dean] Arden, Jack Gilmore, John McCarthy, myself, Arnie Siegel, and [?] who will be in charge of the Lincoln 709

programming to consider using the real time package [IBM's terminology] on the 709 to simultaneously service about 20 Flexowriters in TX-0 fashion. ... We are going to start off with one Flexowriter on the 704 real time package." [R581229-59019] This was the start of MIT's time sharing developments.

In January 1959, "Attended a meeting in Dean Arden's office concerning 704 Flexowriter input ... I didn't feel that the group accomplished very much." [R590112-21]

In early March, "John McCarthy is calling a series of meetings to consider operator and compiler programs for the 7090 computer which will be the transistorized version of the 709 which will replace the 704. We had a meeting yesterday [March 3]." I spoke about MIV and Group Control. "I am hoping that in later sessions [we can show that MIV] should be an integral part of the programmer's bag of tricks applicable to any problem." [R59219-34] But by the end of the month "The meetings to discuss 7090 problems seems to have lost momentum, so it is unlikely we will have another one very soon." [R59323-331]

In April "With the imminent breakup of Whirlwind, we have started to consider putting a good manual intervention console on the 704. I have now decided to soft pedal the connection of TX-0 to 704 until we have a definite use. We got together with John McCarthy and Herb Teager who is now putting the Flexowriter on the 704. [I believe Arnie Siegel, who knew Whirlwind well, had done some early preparations to adapt our earlier work.] John Ward and Don Clements are going to assist in this and it looks like our first move will be to get a light pen hooked up to the 704 to use the present scope ... In the long term, however, we are thinking of developing a self maintaining display system, [Later the ESL Display Console (Ross and Ward 1961, pp. 79-92), which was Rob Stotz's Masters thesis, came from this idea.] possibly built out of Whirlwind components. There is quite a bit of engineering involved there, however. McCarthy and Teager are going over to see the Whirlwind equipment with us at 11 a.m." [R59416-511].

On April 16-18, 1959, a select conference on Symbolic Manipulation (organized by McCarthy, Perlis, and Newell [R59219-34]) was held at MIT. (McCarthy 1959) This was the first public presentation of LISP by McCarthy, COMIT by Yngve, and "I talked for a short time on multi-mode control as applied to list searches, group control, and proposed a modified list structure which seems more appropriate to our design machine application." [R59416-511] This was my "reversed index registers" (as I later called it) method of general pointers and "n-component elements" of Plex Programming, (Ross 1961) which began what now is the field of abstract data types. Evidently Al Newell made no presentation at MIT, but "On May 4, 5 and 6 I attended a conference on self organizing systems ... in Chicago. [Some papers] were quite good, ... [especially] one by Allen Newell on the GPS system general problem solver that he is working on with Simon and Shaw. ... will plan to get together with him." [R59416-511].

On May 12, I took McCarthy, Teager, and Don Clements (then project engineer for the APT Project) "to look over the Whirlwind equipment. They seemed satisfied enough but our present thoughts are to find out what is going on at Lincoln, too." [R59512 & 13] A memo from me (Ross 1959) shows that Arden, Clements, McCarthy, Minsky, Teager, Ward, and Ross visited Lincoln (re 709, Charm, TX-2, and SAGE) on May 19, but "Our meeting out at Lincoln ... was pretty much of a farce. People were late or missing or not very sympathetic ... so not much was accomplished." [R59518-20]

5.6 The Design Machine

So as 1960 rolled around, various seminal equipment developments at MIT had been initiated, but they are better treated as components of the history of time sharing, computer graphics (hardware and software), computer-aided design (CAD), software engineering, and software technology, rather than merely as extensions of the personal work station theme of this paper. I hope to be invited sometime in the future (though not too soon, for these papers are excruciating torture to write!) to contribute to the chronicling of those development, as well.

Any such future consideration of CAD work stations would have to begin with a mention of George Price's prescient article in the November 1956 issue of Fortune Magazine, (Price 1956) and since it, too, was a "first in the fifties", I must reference it here.

Going back again in time, in January 1957 I said

"Bill Webster [Air Force contract monitor for APT] also had a copy of the article on a design machine which I had been told about in New York. It's amazing the similarity between what is in that article and what was in my Gestalt paper and what we have been working on the past year or so. He even has pushbutton language with sentences of the form that we are planning for our milling machine, and the computer drawing pictures of the part being designed on the scope just like our scope plot, etc. I plan to write him a letter describing our work and enclose a copy of my paper and send a copy of this correspondence to the editor of Fortune and see what results. I think it is quite interesting that we have been actually carrying out what in his article is merely a proposal which everybody feels very strongly isn't as wild as it seems. Webster wanted to check with me that his impression that we were doing just what the article called for was correct, and I assured him that it was." [R57017-8]

I finally got a library copy to read by January 25 [R570125], and on February 6 wrote to Price, enclosing a Gestalt paper reprint, and

describing our APT progress, then being issued as the first APT Interim Report. I then go on:

"These routines provide the necessary mathematical structure around which a convenient human language can be built to make a complete automatic programming system. The detailed specification of this human language has not been attacked yet, but it will be the major focus point for the efforts of the group. The language will have specially designed features for description of surfaces and their interrelation, as well as for the instruction of the machine tool itself. The form of the language will probably be similar to that which you propose for the design machine, and as outlined in my paper, although it will be a written language and not use push buttons initially [-- used in the Gestalt paper and in Price's article] ...

"Another important part of the language problem is the computer-to-human language, which we envision in two forms. First there will be the ability of the computer to 'talk back' to the human in the same type of language which the human uses, probably in a written form. The computer's language also will be of a pictorial form in which the computer will draw pictures for the human to check and work from in a way almost identical with your proposals in the FORTUNE article. A rudimentary routine of this sort has been in operation on the Whirlwind I Computer for several months now, which draws on the output oscilloscope of that computer arbitrarily positioned and scaled axonometric projections of the part being made. True perspective will be programmed later if it seems warranted. At present the picture consists of the sequence of 'cut vectors' which are used to program the machine tool, but more elaborate schemes are planned. At the present time the pictorial displays are photographed and no effort has been made to increase the speed of the program so that a cycled display can be viewed easily on the display tube itself. Memotron-type tubes are not presently installed on the Whirlwind Computer although they would be very desirable for this and other applications. [The ARDS Oisplay of Rob Stotz was the first such memory tube display (Ross and Ward 1961, pp. 100-113).]

"The striking similarity between our work and the proposals of your FORTUNE article indicate that we have a number of very strong mutual interests and a similar approach to these problems. Mr. William Webster of the Air Materiel Command, who is our project monitor, brought your article to my attention and showed me a letter which you had written to him. In that letter you mentioned a study which you made concerning a design machine using the IBM 704 computer as a major component. I would very much appreciate a copy of this report if you have any available in order that we may know in more detail your ideas on this subject. We plan to transfer our work to the IBM 704 in the very near future.

"I look forward with anticipation to further correspondence with you on this most interesting subject." [C5726]

A February 25 reply from Price enclosed a reprint of the FORTUNE article and his retyped "supplementary memo", but said he was changing jobs and moving, so he hadn't "been able to find the time to do more than take a quick glance a your two papers." [C57225] On May 13 he visited my home and we "talked for an hour or so ... but we never did get around to talking the same language or really discussing anything ... He has no technical training in this field." [R57513] But he certainly wrote an interesting article, and his Design Machine proposal surely was a first, including many features that later were used in actual CAD work stations. As a side box [p. 153] about "The Author and His Machine" said:

"Price began thinking about his Design Machine some ten years ago. Could the Machine be built today? One leading computer expert shown the proposal was skeptical. Price thereupon prepared a detailed supplementary memorandum demonstrating how an IBM 704 computer could be incorporated into a Design Machine, how a complex part could be described to the Machine, and how with the aid of certain auxiliary devices the Machine would display the part -- in 3-D. Computer experts at IBM pronounced the memorandum eminently sound. The first skeptic conceded that Price had shown how the job might be done."

The main thrust of the article was "eight recipes for modernizing R. and D.," one of which was the Design Machine, and another to mechanize "reading" of technical literature for mechanized filing (with a reference to V. Bush and the Patent Office). Besides describing the pushbutton and joy stick preparation of stereoscopic pictures of parts and "invisible models -- recorded in a magnetic memory," with "preparation of control tapes for guiding automatized machine tools," the article says "a single Machine might be shared among several companies scattered around the country, for a number of control stations can be in use simultaneously, with keyboard signals being briefly stored on magnetic tape, and the central computer switching from one station to another every few seconds." [p. 228] [-- an idea also promoted by Jay Forrester for Whirlwind, in 1948 (Redmond and Smith 1980, pp. 233, 234).] Price's article was well ahead of its time.

Section 6

SYSTEM SOFTWARE

As I said at the beginning, a final essential ingredient for a viable, general, PWS scheme is a systematized way to link the work station capabilities to the class of problems to be solved through some form of software framework. As I have indicated, the earliest such framework, when Whirlwind had only 2K words of memory, took the form of several programming principles, such as "store switch values in variables", the language design principles of the Gestalt Programming paper, and the rudiments

of drum/ES storage management prompted by the Mistake Diagnosis Routine (MDR). But with the advent of core memory, and its expansion to 6K words, and with the growing sophistication of approach engendered by early efforts at MIV in the Evaluation Program, a truly systematized solution was the natural evolutionary step.

6.1 The Servo Lab Utility Routine Program

The first coherent expression of these ideas appeared in (Ross 1956h) [note -- when APT work was just getting under way, as well], which said:

"After some experience with the integration of the manual intervention facilities into the previously fully automatic evaluation programs, by the Servomechanisms Laboratory, it was realized that the manual intervention features could be used as a direct aid to programming by incorporating a number of general-purpose utility routines into the system. Shortly thereafter, the new task concerning the investigation of the techniques and problems for an airborne mission director was undertaken. Since detailed specifications on the mission director problem were not then available, and also since the committment of the evaluation studies to the B-58 program made progress in that direction imperative, it was decided that a worthwhile merging of these two interests could be achieved. This was done by commencing the elaboration of the evaluation program on the Whirlwind computer into a general purpose problem-solving system combining the talents of the human programmer and the electronic computer for the solution of general computer problems. The result is a human-computer system which may be viewed either as an elaborate evaluation program, or as a prototype problem-solving program with the problem of evaluation of airborne fire-control systems as the motivating core. This memorandum stresses the latter viewpoint. [p. 1] ...

"The combined evaluation and problem-solving system which is the subject of this memorandum has been given the name, the Servo Lab Manual Intervention System (MIV System). The main feature which distinguishes this system from other computer programming systems is that all of the major facilities of the system are instantaneously available under manual intervention control. [p. 3] ...

"The facilities of the MIV System may be grouped into three categories according to their primary function and motivation. Some of the features are concerned with the problem itself (in the present case the evaluation program). Other features are strictly of a utility nature having to do with input and output of data in various tabulated and graphical forms. The third category concerns features which arise almost entirely from the fact that a manual intervention system is being used and have primarily to do with reliability and ease of operation of the overall system. The main features of each of these categories will be described in turn." [p. 3]

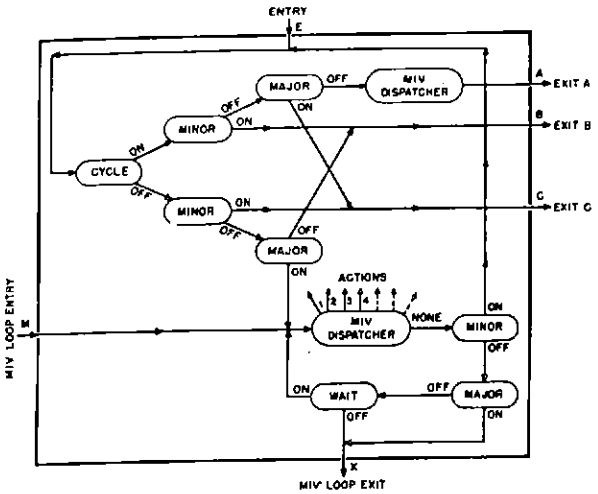
Five more pages then detail the plans for what came to be called "SLURP", just nine days later, in the second Computer Applications Group memo, mentioned earlier. (Ross 1956i) The one-page summary of features, some of which were implemented only in later years, was:

"The Servo Lab utility routine program (SLURP) is a system of Whirlwind Computer programs which combine the manual intervention features of that computer, with modern programming techniques, into a unique problem-solving system. SLURP includes all of the facilities of the Comprehensive System developed for the Whirlwind Computer by the Digital Computer Laboratory staff, plus a large number of general purpose routines for extracting and analyzing information about arbitrary computer programs. The governing philosophy of the system is to allow the programmer to work on one small part of a very large and complicated problem, virtually independent of the computer which is being used and those aspects of the overall problem which are not of immediate concern. The major features of the system which make this possible are as follows:

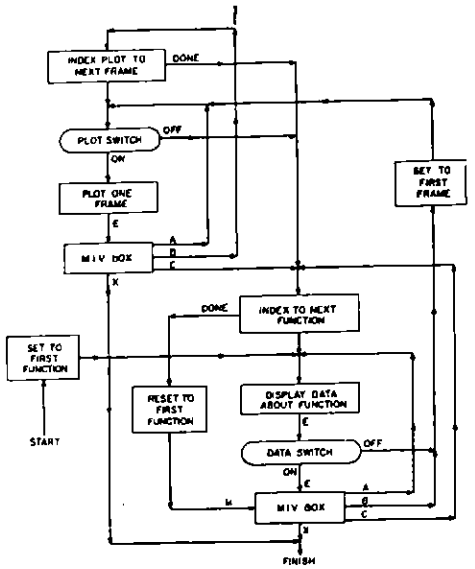
"1. A group control program which allows the automatic incorporation of new sections of programming into the system.
"2. A manual intervention (MIV) system of programs which allow the programmer to interrogate and instruct the computer with respect to the overall problem in terms of a specially designed and easy-to-use language represented by: switches, buttons, lights, and visual displays.
"3. A logging program which records all of the MIV actions taken by the programmer.
"4. An editing program which edits the logged information into an easy-to-read record of the manual actions taken, and generates a log playback tape.
"5. A log playback program which automatically simulates manual actions in response to the instructions on the log playback tape.
"6. An elaborate set of plotting and tabulating programs for data presentation and record keeping.
"7. A mistake diagnosis routine (MDR) which may be used to abstract arbitrary intermediate computed quantities from any program and present these quantities in any of the above forms [including scope].
"8. The SLURP program proper which is a simulated, generalized, special-purpose computer which allows the incorporation of all of these routines into a smoothly functioning system, along with the capacity for continued expansion of these facilities." [p. 3]

6.2 The SLURP Simulated Computer

The SLURP computer had many features that independently were discovered and incorporated into the Burroughs B5000 computer (I was astounded to see the similarities when I first



Definition of MIV Box re-entrant subroutine of Servo Lab Utility Routine Program (SLURP) simulated computer — basis for systematized MIV languages.



Hierarchic control of complex program action with generic MIV Box controls. Precussor to current mouse/menu control scheme.

read their manuals in the early 1960s). SLURP included built-in Group Control (memory paging), the Memory Table (virtual addresses for re-entrant program segments), Alarm Conditions (exception conditions with optional MIV and MDR facilities), and the ZIP Interpretive Program (which "serves the important function of establishing the MIV language which is appropriate for a particular program" -- and integrated all of the other features, including I/O in a powerful, very condensed macro-command-like language).

A unique feature of SLURP was its "multi-mode control element". (Ross 1958) As the definitive report on SLURP says:

"The instruction repertoire of the SLURP computer contains no mathematical instructions but instead concentrates on a wide variety of a) jump instructions for transferring control from one sequence of instructions to another, and b) instructions which control selection and adjustment of input-output equipment. Individual flexowriter typewriter characters are included in the instruction repertoire and the execution of this type of instruction involves the reproduction of these characters on a selected output device. A SLURP instruction may occupy an arbitrary number of Whirlwind registers. There are two binary digits called the A and C bits which are set aside in each instruction and are used to label individual instructions for execution in the various modes of the control element ...

"If the A bit is ONE and the C bit ZERO, the instruction will be executed when the control element is set to the A mode but will not be executed if the control element is set to the C mode [etc.]. All SLURP instructions (with the exception of a few jump instructions) are executed when the control element is set to the B mode, independent of the settings of the A and C bits. Since the mode of the control element may be set by jump instructions, the setting of the control element may be changed as often as necessary to accomplish a desired result ...

"In addition to the return jump instruction for termination of a remember jump sequence, it is also possible to remember jump to a flexo phrase (any sequence of characters or words), and the occurrence of a stop character in that flexo phrase will act as a return jump under certain circumstances. However, if that same flexo phrase is executed by encountering it in the normal sequence of operations (instead of by an entry via a remember jump), the stop character is ignored. In this way a lengthy flexo phrase which contains useful subphrases or words may be extracted from the entire sequence by means of remember jump instructions." [pp. 3-82 to 3-84]

This feature was used to have abbreviated labels for display and long labels for printouts, and the same technique gave complex behavior if the "words" were other instructions. Our favorite test case

```
THE SKUNK SAT ON THE STUMP
THE SKUNK THUNK THE STUMP STUNK
THE STUMP THUNK THE SKUNK STUNK
```

took only 13 SLURP instructions including output device selection. [p. 3-85] SLURP packed a tremendous punch in very little space.

6.3 The MIV Box

One of the most powerful features of MIV language design with SLURP was the MIV Box, a re-entrant subroutine that could be used in many places. [p. 3-127 to 3-137] The basic language idea was that "if the human does not like what is presently going on, he merely needs to remember to depress an appropriate exit button. The appropriateness of the two exit buttons can be given a universal meaning by referring to one as a major exit, and the other as a minor exit. Then no matter what program is operating, if the human is unsure which of the two exit buttons to press, he may first try the minor exit and if that does not give the desired action, then pressing the major exit is guaranteed to work. ... It is not necessary to know beforehand all the possible actions which may be controlled by exit buttons if it can be established that those programs will operate on a hierarchy of actions. The major and minor exits can then be associated merely with a change in level within whatever hierarchy exists for a particular program. This uniformity of meaning of language is achieved by combining the MIV Dispatcher, the wait switch, the major and minor exit buttons, and one new switch, the 'cycle' switch." The example shows how only two more switches allowed many choices of control for selecting and plotting successive frames of a sequence of functions, as in the Evaluation Program. One more switch could similarly specify console display only or off-line camera, as well, and the SLURP system would efficiently see to the details.

CONLCUSION

I hope that I have succeeded in defending my thesis that there were indeed some significant PWS firsts in the Fifties, even though only a tiny fraction of the computer resources that today are thought to be essential were then available. The revolutionary evolution since those early days has indeed brought much greater sophistication and broader capabilities -- including the all-important hardware, software, and systematized approach improvements which allow ubiquitous spread to all types of users. But don't sell the old days short. Big ideas can come in small packages. Maybe that's an idea we've lost track of in today's technology where we shrink things mechanically to make them smaller. Maybe we need a sharper return to those earlier days when the only recourse was to think big things into their distilled essence -- and make them work.

THANKS

... to the MITRE Corporation Archives and the MIT Museum for the Whirlwind photos.

... to MIT and my many colleagues for making it all possible.

REFERENCES

AMERICAN ORDNANCE ASSOCIATION (1955) April 14-15. Symposium on Proving Ground Instrumentation Problems, Air Force Missile Test Center, Patrick Air Force Base, Cocoa, Florida (program and papers).

BRACKETT, C.F. (1959) February 2. The Direct Flexowriter Keyboard Input System at Whirlwind I (Barta). Lexington, MA: Division 2 - Lincoln Lab. Interoffice Memo (no number), 3 pp.

CHARACTRON PROJECT (1955). Charactron Display Console Model 70B (January 24, 1955, 14 pp.); Charactron Computer Readout System, Model 100 (June 3, 1954, 22 pp.); The Charactron Rapromatic (no date, 11 pp.). San Diego, CA: Convair division of General Dynamics Corp.

COHEN, ARNOLD (1985) October 4. Oral comments to D.T. Ross at Annual Meeting of the Charles Babbage Foundation.

CONVAIR (1954) August 9. Agenda for Flight Test Program Meeting see also Ross (1954).

EJCC (1955) November 7-9. Program of the Eastern Joint Computer Conference and Exhibition, Statler Hotel, Boston, 27 pp.

ELECTRONIC DESIGN MAGAZINE (1956) May 1. New York, NY: Hayden Publishing Co. Computer Developments: Design Trends from Meetings, pp. 36 & 37.

ERA (1954a). ERA 1103 General Purpose Computer System (brochure). New York, NY: Engineering Research Associates Division of Remington Rand, Inc., 4 pp.

ERA (1954b) January 5. ERA 1103 Installation Requirements, St. Paul, MN: Engineering Research Associates Division of Remington Rand, Inc., 16 pp.

FORRESTER, J.W. (1948) July 29. Whirlwind High Speed Computing. Cambridge, MA: MIT Servo Lab. Rep. No. R-142. Appendix A in Redmond and Smith (1980), pp. 225-236.

HAMILTON, D.A. (1955) August 4. Conventions for Presentation of Results from Whirlwind I Polynomial Fit Program. Cambridge, MA: MIT Servo Lab. Memo 7138-M-116, 4 pp. See also Ross (1954), and Ross Thompson, and Cundiff (1956).

HOLMES, L.L. (1956) April 3. Flexowriter Input to WWI. Lexington, MA: Division 6 - Lincoln Lab. interoffice memo, 2 pp.

McCARTHY, J. (1959) February 27. Invitation to Conference on Symbol Manipulation Systems, April 16-18, MIT.

MIT DIGITAL COMPUTER LAB. (1956) March 1 and 2. Seminar on Micrprogramming (Announcement and agenda). Cambridge, MA. 2 pp. plus map. June 6, 1956: Partial transcript of D.T. Ross presentation pp. 152-161, via F.W. Helwig.

MIT SERVD LAB. (1952-1958). Certain Lists of laboratory personnel.

MIT SERVO LAB. (1955) March 8, 9. Symposium on Design and Evaluation of Bomber Fire-Control Systems.

MIT SERVO LAB. (1958) December. Data Reduction Programming for Pre-B-58 Tests of the XMD-7 Fire-Control System (3 Volumes). Cambridge, MA: MIT Servo Lab. Reports: Ross, D.T. Functional Description of the Data Reduction System (7886-R-1), 44 pp; Scheff, B.H. Initial Data Processing (7886-R-2), 149 pp.; Ross, D.T. and McAvinn, D.F. Evaluation of Fire-Control System Accuracy, (7886-R-3), 177 pp.

NIELSEN, K.L. and HEYDA, J.F. (1951) July 20. Mathematical Theory of Airborne Fire Control. Washington, D.C.: U.S. Government Printing Office, NAVORD Report 1493, 208 pp.

PRICE, G.R. (1956) November. How to Speed Up Invention. New York, NY: Fortune magazine, p. 150 ff.

PRICE, G.R. (1957) February. Technical Details Concerning the Design Machine. Kingston, NY: personal memo, 72 pp.

PROJECT WHIRLWIND (1952-1957) Quarterly Summary Reports No. 31 - No. 50. Cambridge, MA: MIT Digital Computer Lab.

REDMOND, K.C. and SMITH, T.M. (1980). Project Whirlwind: The History of a Pioneer Computer. Bedford, MA, Digital Press, 280 pp. Index.

ROSS, D.T. (1952) July. Autocorrelation Program working papers.

ROSS, D.T. (1953a). First Evaluation Program Flow Diagram. Computation Notebooke pp. 26 & 27.

ROSS, D.T. (1953b) June 27. An Introduction to the Use of Air-Mass Ballistic Tables in Airborne Fire-Control System Evaluation (CONFIDENTIAL, Title Unclassified -- declassified November 1958). Cambridge, MA: MIT Servo Lab. Rep. No. 6506-ER-46, 25 pp.

ROSS, D.T. (1953c) November 24. A Mistake Diagnosis Routine for Whirlwind I Programs. Cambridge, MA: MIT Servo Lab. Rep. No. 7138-R-1, 26 pp.

ROSS, D.T. (1954a) May 24. Missile Launch Simulator program for Memory Test Computer working papers.

ROSS, D.T. (1954b) June 25. Improved Computational Techniques for Fourier Transformation. (MS Thesis). Cambridge, MA: MIT Servo Lab. Rep. No. 7138-R-5, 84 pp.

ROSS, D.T. (1954c) July 16. Letter to W.J. Moe of Engineering Research Associates (ERA) regarding 1953 Polynomial Fit Program, 2 pp. See also Hamilton (1955).

ROSS, D.T. (1954d) August. Scope Input Program (no original; revised copy fc Tape 126-50-505, 2/15/55).

ROSS, D.T. (1955a) March 8. Handwritten instructions to DAH for ROSS DEMO. Director Tape, plus printout of same.

ROSS, D.T. (1955b) June. A Basic Input Translation Program for the ERA 1103 (program, runs, and working papers).

ROSS, D.T. (1955c) July 25. Special In-Out Equipment for Human Participation in High-Speed Computer Operation. Cambridge, MA: MIT Servo Lab. Memo 7138-M-112, 8 pp.

ROSS, D.T. (1955d) October 13. [The] Gestalt [System of] Programming -- A New Concept in Automatic Programming, handwritten draft, 15 pp.

ROSS, D.T. (1955e) November-December. Handwritten draft pages.

ROSS, D.T. (1955f) November 17. Gestalt Programming: A New Concept in Automatic Programming (second draft, typed) 12 pp. plus addendum: "Proposed Changes for Final Draft", 2 pp.

ROSS, D.T. (1955g) December 14. Proposed Equipment Modifications for the Pre-B-58 Data Reduction Programs. Cambridge, MA: MIT Servo Lab. memo 7138-M-128, 7 pp.

ROSS, D.T. (1956a) January. Handwritten oral presentation drafts and notes.

ROSS, D.T. (1956b) February 7. Gestalt Programming: A New Concept in Automatic Programming. New York: Proceedings of the Western Joint Computer Conference. AIEE for the JCC (now AFIPS), pp. 5-10. Also Cambridge, MA: MIT Servo Lab. Rep. No. 7138-TM-7, 14 pp.

ROSS, D.T. (1956c) March 1. Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-144, 6 pp.

ROSS, D.T. (1956d) March 14. Flexowriter Keyboard Input to WWI, Preliminary Specifications. Cambridge, MA: MIT Servo Lab. Memo 7138-M-148, 3 pp.

ROSS, D.T. (1956e) April 5 - Summer 1956. Proposed Microprogramming Instruction for Whirlwind I Computer. Working papers, drawings, and memo outline.

ROSS, D.T. (1956f) April 30. A Proposed Flexowriter Input for the Whirlwind I Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-160, 4 pp.

ROSS, D.T. (1956g) September 26. Servomechanisms Laboratory Requirements for Computing Facilities, 1956-1957. Cambridge, MA: MIT Servo Lab. memo CA-M-1, 4 pp.

ROSS, D.T. (1956h) November 14. Programming Progress for Mission Director Task. Cambridge, MA: MIT Servo Lab. memo 7138-M-200, 8 pp.

ROSS, D.T. (1956i) November 23. Whirlwind Versus 704 for Servo Lab Problems. Cambridge, MA: MIT Servo Lab. memo CA-M-2, 5 pp.

ROSS, D.T. (1958) April. A Multi-Mode Control Element; A Philosophy of Problem Solving; The SLURP System for Experimental Programming. Cambridge, MA: Research in Defense Techniques for Airborne Weapons: 1957 Annual Report, Volume 2. MIT Servo Lab. Rep. No. 7668-R-5(2), pp. 3-81 through 3-163.

ROSS, D.T. (1959) May 13. Visit to Lincoln Lab., Tuesday, May 19. Cambridge, MA: MIT Servo Lab. memo (no number).

ROSS, D.T. (1961) March. A Generalized Technique for Symbol Manipulation and Numerical Calculation. Communications of the ACM 4(3): 147-150.

ROSS, D.T. (1977). Origins of the APT Language for Automatically Programmed Tools. New York: History of Programing Languages (R.L. Wexelblat, Ed.), (1981) Academic Press, pp. 279-367.

ROSS, D.T., Thompson, D.A. (Hamilton), and Cundiff, T. (1956) December 17. Polynomial Fit Program. Cambridge, MA: MIT Servo Lab. Memo 7138-M-202, 18 pp. See also Hamilton (1955).

ROSS, D.T. and Ward, J.E. (1956) May. Investigations in Computer-Aided Design for Numerically Controlled Production (Final Report for 1 December 1959 - 3 May 1967). Cambridge, MA: MIT Electronic Systems Lab. for Air Force Materials Lab., WPAFB, Ohio. Rep. No. AFML-TR-68-206. MIT Rep. No. ESL-FR-351, 229 pp.

ROSS, D.T. and Ward, J.E. (1957) October 22. Charactron Plot Program for Pre-B-58 Data Reduction. Cambridge, MA: MIT Servo Lab. memo 7668-M-251, 4 pp.

ROSS, D.T. and Ward, J.E. (1961) January. Picture and Pushbutton Languages (Chapter VIII), and Ward, J.E. A Manual Intervention Facility (Chapter IX), in Investigations in Computer-Aided Design, Interim Report for December, 1959 to May 30, 1960. Cambridge, MA: MIT Electronic Systems Lab. Rep. No. 8436-IR-1, 156 pp.

WARD, J.E. (1954) July 23. Disposal of Mechanical Correlation Computer (MCC). Cambridge, MA: MIT Servo Lab. Memo 7138-M-49, 2 pp.

WARD, J.E. (1955) May 2. Symposium on Data Reduction (Announcement). MIT Servolab (no number).

WARD, J.E. (1958) January 27. Manual Intervention. (Memo to D.T. Ross.) Cambridge, MA: MIT Servo Lab. (no number).

WARD, J.E. (1960) January 15. Automatic Programming of Numerically Controlled Machine Tools (Final Report). Cambridge, MA: MIT Electronic Systems Lab. Rep. No. 6873-FR-3, for June 26, 1956 to November 30, 1959. Reprints 6873-FR-1 (July 30, 1952) for July 1949 to July 1954, [sic] and reprints 6873-FR-2 (March 15, 1956) for July 1949 to March 15, 1956.

WARD, J.E. (1970) June. Computer-Aided Design for Numerically Controlled Production (Final Report for 1 May 1967 - 30 January 1970). Cambridge, MA: MIT Electronic Systems Lab. for Air Force Materials Lab., WPAFB, Ohio. Rep. No. AFML-TR-70-78. MIT Rep. No. ESL-FR-420, 121 pp.

WARD, J.E. and Ross, D.T. (1955) July 29. Minutes of M.I.T. Oata Reduction Symposium: June 1 and 2, 1955. Cambridge, MA: MIT Servo Lab Rep. No. 7138-S-1, 20 pp.

WARD, J.E. and Ross, O.T. (1956) March 1. Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-144A, 30+ pp.

WARD, J.E. and Ross, D.T. (1956) May 8. Draft paper of Revision of (Ross 1956c), incorporating MIT Digital Computer Lab. memo and drawings. Cambridge, MA: MIT Servo Lab. memo 7138-M-144A.

WJCC (1956) February 7-9. Program of the Western Joint Computer Conference and Exhibit. Fairmont Hotel, San Francisco, 23 pp.

WOLF, W.M. (1954) September-November. Worksheets and photos for Large Letter program. Flow diagram and WWI code for Manual Intervention tests.

Some Reflections on Early History

J. C. R. Licklider
Massachusetts Institute of Technology

Abstract

The ideas and developments reviewed briefly are mainly from six sources: (1) the M.I.T. Lincoln Laboratory during the middle and late 1950s, (2) Bolt Beranek and Newman in 1957-1962, (3) the ARPA program in 1962-1964, (4) IBM Research in 1964-1966, (5) Ivan Sutherland's lab at Harvard in 1967, and (6) Project MAC at M.I.T. in 1968-1972.

For instance, there were (1) cloud filter stations with light guns and light pens, (2) a single 1024x1024 scope display time-shared four ways and a workstation for studying technical documents, (3) Rand tablets, Teagher tables, and Bernstein's horizontal text-editing display onto which both the human editor and the computer editor could write (and understand each other), (4) a computer system especially for computer-based education and a powerful vector graphics display system, (5) a PDP-1-based, single-user workstation with four display screens, a paper tape unit, a magnetic tape unit, a printer, and the beginnings of Sutherland's virtual space displays, and (6) graphics over the ARPANET and a multi-window workstation based on the Evans and Sutherland LDS-1.

One of the key insights to be offered explains why workstations have vertical instead of horizontal (desk-like) display surfaces. Part of the plot is that, when, in the 50s or 60s, a cathode-ray tube was mounted face-up, some of the phosphor fell down onto the electron gun and caused a short circuit. Another part is that Fredkin and DEC simply got mixed up about which way the PDP-1 display ought to swivvel.

Imitation Issues – what discuss?
Instrumentation
Input devices
output devices { quality, size
Software for users; learnability
Hudson Lab
Workstation DC Unix
Documentation as part of pws
SW with group property; convenience of use

system for studying stuff

The ARPANET & Computer Networks

Lawrence G. Roberts
NetExpress Inc.

In 1964 only large mainframe computers existed, each with its own separate set of users. If you were lucky the computer was timeshared, but even then you could not go far away since the terminals were hardwired to it or connected by local phone line. Moreover, if you wanted data from another computer you moved it by tape and you could forget wanting software from a different type of computer. Thus, most users were tied by their computer and terminal to a very restricted enviornment.

Today, in 1985, your terminal could well be a microcomputer networked with a very large, worldwide collection of other computers. You can obtain data and software from all these computers relatively easily with room for improvement) or, where convenient, use the software and data on its home computer by remote access, computer to computer.

This change, which has occurred over the past 20 years, is in part a massive and evolutionary change in computer technology, and in part a modest and revolutionary change in communications technology. The revolution in communications started with an experiment in computer networking, the ARPANET, and grew into a communications revolution called packet switching. Today virtually all the world is linked by packet switched communications service so that any terminal can access almost any computer in the world. This packet switched data network has grown up independent of the telephone network, but over the next 20 years the basic fabric supporting all switched services (data, telephone and video) appear likely to become converted to packet switching, completing the revolution.

History of Network Concepts

Going back to examine the history of computer networks, the first event for me took place in November 1964 at the Second Congress on the Information System Sciences in Hot Springs, Virginia. In informal discussions with J. C. R. Licklider, F. Corbato, and A. Perlis, I concluded that the most important problem in the computer field before us at that time was computer networking; the ability to access one computer from another easily and economically to permit resource sharing. That was a topic in which Licklider was very interested and his enthusiasm infected me. My interest was more toward the networking and communications issues rather than the computer language and compatability issues which were foremost in Lick's mind. For at least the prior year, Licklider, who was then running the ARPA IPT Office (then called Command & Control Research), had been pursuing the concept of the "Intergalactic Computer Network," trying to define the problems and benefits resulting from computer networking. In any case, that Hot Springs discussion convinced me that I should change my carrier objectives to concentrate on computer networking and the related communications problems.

One year later, in 1965, a second important meeting took place at MIT. Donald Davies from the National Physical Laboratory in the U.K. was at MIT to give a seminar on time sharing. Licklider, Davies, and I discussed networking and the inadequacy of data communication facilities for both time sharing and networking. Davies reports that shortly after this meeting he was struck with the concept that a store and forward system for very short messages (now called packet switching) was the ideal communication system for interactive systems. His wrote about his ideas in a document entitled "Proposal for Development of a National Communication Service for On-Line Data Processing" which envisioned a communications network using trunk lines from 100K bits/sec in speed to 1.5 megabits/sec (T1), message sizes of 128 bytes and a switch which could handle up to 10,000 messages/sec (Historical note: this took 20 years to accomplish). Then in June 1966, Davies wrote a second internal paper, "Proposal for a Digital Communication Network" in which he coined the word **packet**, a small sub part of the message the user wants to send, and also introduced the concept of an "interface computer" to sit between the user equipment and the packet network. His design also included the concept of a Packet Assembler and Disassembler (PAD) to interface character terminals, today a common element of most packet networks.

As a result of distributing his 1965 paper, Donald Davies was given a copy of an

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1986 ACM-0-89791-176-8-1/86-0051 $00.75

internal Rand report, "On Distributed Communications" by Paul Baran of the Rand Corporation, which had been written in August 1964 (1). Baran's historical paper also described a short message switching network using T1 trunks and a 128 byte message size but was oriented toward providing extremely reliable communications for secure voice and data in a military environment. In all, there were 11 reports written for the Air Force in the Rand Memorandum group, of which a couple were classified and unfortunately the others were very sparcely published in the scientific press, thus their impact on the actual development of packet switching was mainly supportive, not sparking its development - that happened independently at Rand, NPL and ARPA.

The First Network Experiment

Convinced that computer networking was important, the first task was to set up a test environment to determine where the problems were. Thus, in 1966, I set up two computer networks between Lincoln Laboratory's TX-2 computer and System Development Corporation's Q-32 computer using a 1200 bps dial channel (high speed those days). Each computer was operating in time-sharing mode and permitted any program to dial the other computer, log-in, and run programs much as it would execute a subroutine call. The experiment showed that there was no problem getting the computers to talk to each other and use resources on the other computer - time-sharing operating systems made that easy. The real problem uncovered was that dial communications based on the telephone network were too slow and unreliable to be operationally useful. This work, jointly authored with Tom Marill, was published in the AFIPS FJCC proceedings, Nov. 1966 (2). The lesson learned - **a new data communications network is needed in order to successfully network computers.**

ARPANET Development

The chance to develop and build a major computer network experiment based on radically new communications technology came within a few months. I was asked to take over the responsibility of the ARPA Information Processing Techniques (IPT) office and manage and build its programs. ARPA was sponsoring computer research at leading universities and research labs in the U.S. These projects and their computers provided an ideal environment for an experimental network project; consequently, the ARPANET was planned during 1967 with the aid of these researchers to link these projects' computers together. One task was to develop an computer interface protocol acceptable to all 16 research groups. A second task was to design a new communications network technology to support 35 computers at 16 sites with 500,000 packets/day traffic. The initial plan for the ARPANET was published in October 1967 at the ACM Symposium on Operating System Principles in Gatlinburg Tennessee (3). The reasons given at that time for establishing a computer network were:

A. Load Sharing: Send program and data to remote computer to balance load.
B. Message Service: Electronic mail service (mailbox service).
C. Data Sharing: Remote access to data bases.
D. Program Sharing: Send data, program remote, e.g. Supercomputer.
E. Remote Service: Log-in to remote computer, use its programs and data.

The communications network design was that of the now conventional packet network; Interface Message Processors (IMP's) at each node inteconnected by leased telecommunication lines providing a store and forward service on very short messages. The main difference from later packet nets was that the IMP's were located at the computer sites and connected by a short parallel cable rather than a communications line interface.

Also presented at the Gatlinburg Symposium was Donald Davies's first open publication on the NPL packet network concepts presented by Roger Scantlebury, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals." (4) It detailed the concept of a high level packet net with high capacity nodal switches and interface computers in front of mainframe computers. This was the first time that either Davies or I knew anything about the work of each other since our 1965 contact. The NPL paper clearly impacted the ARPANET in several ways. The name "packet" was adopted, much higher speed was selected (50 Kilobit/sec vs. 2.4 Kilobit/sec) for internode lines to reduce delay and generally the NPL analysis helped confirm the concept of packet switching.

Another confirmation of the basic concepts came from finally being able to read the Rand reports on distributed communications. The Rand work was very detailed, since it covered the whole network including microwave and one valuable analysis on routing. Their hot-potato routing algorithum was a useful starting point for the ARPANET routing design.

During 1968, a request for proposal was let for the ARPANET packet switching IMP equipment and the operation of the packet network. The RFP was awarded to Bolt Beranek and Newman Inc. of Cambridge, Massachussetts, in in January 1969. The RFP specified the general packetswitching concept, packet size, and interface protocol so that bidders could not totally change the system concept, to circuit or message switching for example. The two largest computer companies to receive the RFP no bid it because they didnt have mini-computers with which to make an economic bid. BBN bid the Honywell 516 mini-computer which was ideal for the task in 1969. Significant aspects of the network's internal operation, such as routing, flow control, software design, and network control were developed by a BBN team consisting of Frank Heart, Robert Kahn, Severo Ornstein, William Crowther, and David Walden. By December 1969 four nodes of the net had been installed and were operating effectively.

The first set of detailed papers covering the ARPANET were published in May 1970 at the AFIPS SJCC. (5-9) These papers reported the motivation and economics (5), the detailed design of the IMP (6), the network delay analysis and experience (7), the topological design programs and results (8), and the host-to-host protocol (9). These papers showed the world for the first time that **packet switching works**, that it is economic and that it is **reliable** and **virtually error free**. They also provided a complete description of how a working network was designed. As such these papers were the technical and motivational basis for many other network experiments around the world.

The ARPANET utilized minicomputers at every node to be served by the network, interconnected in a fully distributed fashion by 50 KB leased lines. Each minicomputer took blocks of data from the computers and terminals connected to it, subdivided them into 128 byte packets, and added a header specifying destination and source addresses; then, based on a dynamically updated routing table the minicomputer sent the packet over whichever free line was currently the fastest route toward the destination. Upon receiving a packet, the next minicomputer would acknowledge it and repeat the routing process independently. Thus, one important characteristic of the ARPANET was its completely distributed, dynamic routing algorithm on a packet-by-packet basis, based on a continuous evaluation within the network of the least delay paths, considering both line availability and queue lengths.

The technical and operational success of the ARPANET quickly demonstrated to a generally skeptical world that packet switching could be organized to provide an efficient and highly responsive interactive data communications facility. Fears that packets would loop forever and that very large buffer pools would be required were quickly allayed. Since the ARPANET was a public project connecting many major universities and research institutions, the implementation and performance details were widely published. (10, 11, 12, 13, 14). The work of Leonard Kleinrock and his associates at UCLA on the theory and measurement of the ARPANET has been of particular importance in providing a firm theoretical and practical understanding about the performance of packet networks.

The ARPANET was first demonstrated publicly at the first International Conference on Computer Communications (ICCC) in Washington, D.C. in October 1972. Robert Kahn of BBN organized the demonstration installing a complete ARPANET node at the conference hotel, with about 40 active terminals permitting access to dozens of computers all over the U. S. This public demonstration was, for many (if not most) of the ICCC attendees, proof that packet switching really worked. At this time, it was difficult for many experienced professionals to accept the fact that a collection of computers, wideband circuits, and minicomputer switching nodes (equipment totaling well over 100 pieces) could all function together reliably. The ARPANET demonstration lasted for three days and clearly displayed its reliable operation in public. The network provided highly reliable service to thousands of attendees during the entire duration of the conference.

Industry Reception

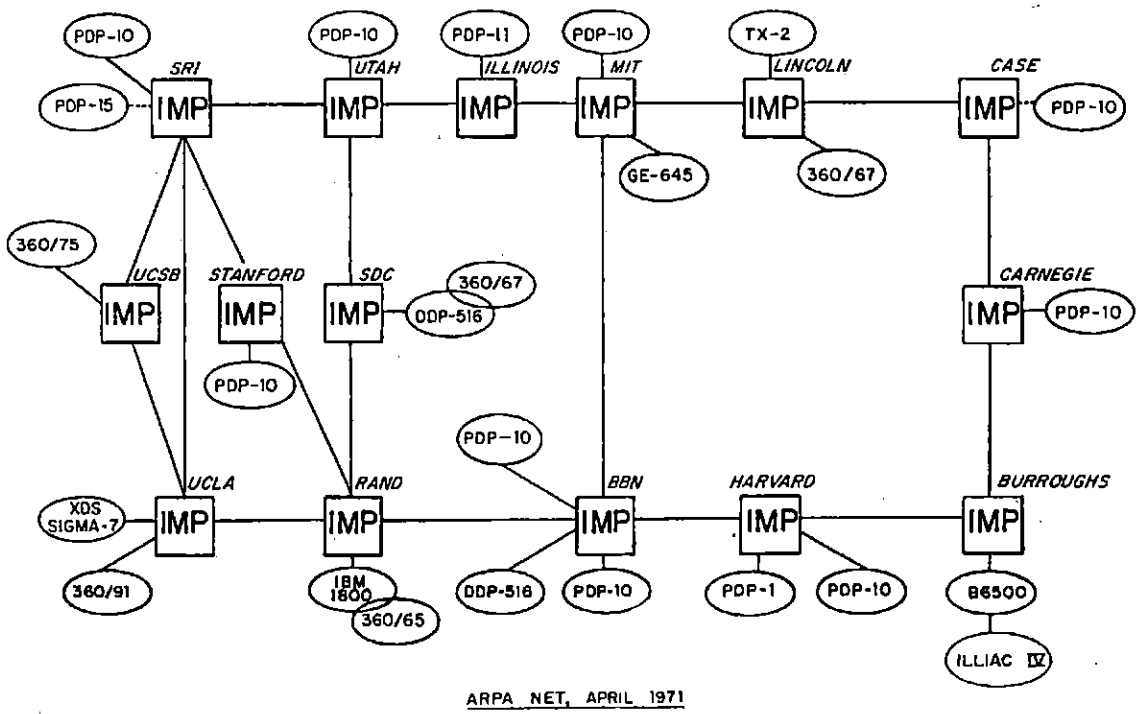
From the first time I distributed a description of packet switching outside the computer research community (the 1967 paper) until about 1975, the communication industry's reaction was generally negative since this was such a radically different approach. In some of the initial technical speeches I gave, communications professionals reacted with considerable anger and hostility, usually saying I did not know what I was talking about since I did not know all their jargon vocabulary. The most common technical flaw suggested (before the ARPANET was built) was that the buffers would quickly and catastrophically run out. After the ARPANET was operating successfully, their pitch changed to be that packet switching would never be economic without the government subsidy. Paul Baran reported the same reaction to his papers when he presented them; this reaction was the major reason his proposals never moved the military. Donald Davies reported a somewhat less angry response from the British Post Office, more one of mild interest but no serious consideration.

I learned a major lesson from that experience; **people hate to change the basic postulates upon which considerable knowledge has been built**. In the case of packet switching, the first postulate to change was the statistical nature of the traffic - data versus voice. The second was that computing was expensive. Some people find it is impossible to consider such a major jolt to their memory organization - they avoid it with putdowns if possible, if not with anger. Other people are more willing to reconsider, but for everyone it requires considerable effort. Those of us proposing packet switching all came from the computing field and did not need to change lots of prior concepts and knowledge. Many of those in the communications field still have not accepted packet switching. (The fight is heating up again as voice packet switching starts to be considered.)

ARPANET Growth

As soon as the first four nodes were brought up and tested in December 1969 the network grew very rapidly. One year later, in December 1970, the network had grown to 10 nodes and 19 host computers. By April 1971, there were 15 nodes with 23 host computers. The topology was as shown in Fig. 1. By this time, it was clear that connecting terminals directly to the network through a PAD-type device was important. Such a device was designed and built in 1970/1971, and the first Terminal Interface Procesor (TIP) was added

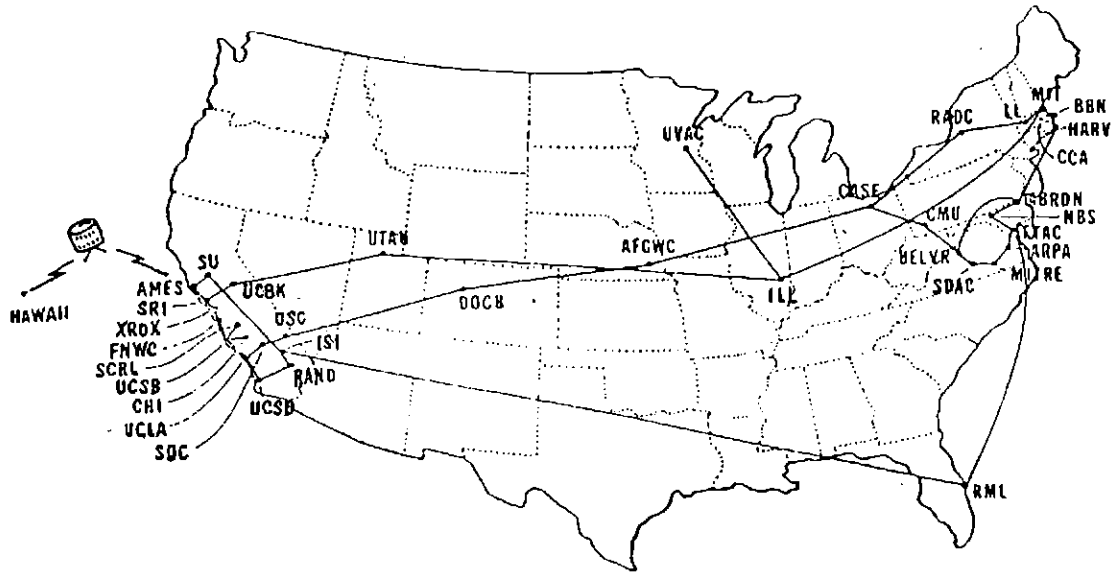
FIGURE 1



ARPA NET, APRIL 1971

FIGURE 2

ARPA NETWORK - JAN 1973



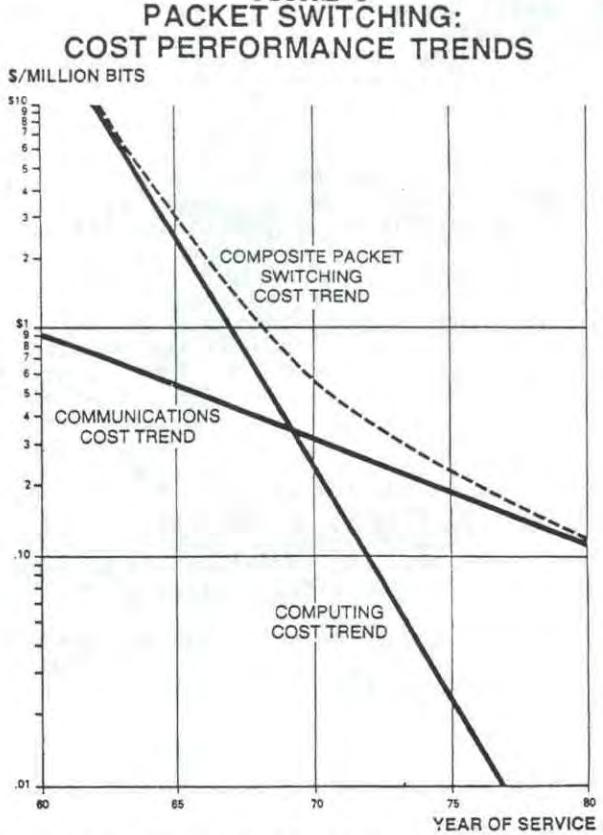
to the network in Aug 1971. This permitted users with no computer to select a computer from all those around the country. In many cases having the user attach his terminal to a TIP and access even his own host(s) through the network was found to be more reliable. This was the start of a trend which today is almost the rule, **workstations should attach to a network, not a computer**. By January 1973, the network had grown to 35 nodes of which 15 were TIP's and was connected to 38 host computers. Fig. 2 shows a map of the network at this point. Network traffic had grown rapidly in 1972, from 100,000 packets/day to 1,000,000 packets/day, exceeding the original estimates I made in 1967. Also in 1973, the first satellite link was added to the network with a TIP in Hawaii; later in the year a pair of TIP's were added in Europe. In September 1973, the network was up to 40 nodes, 45 host computers with internode traffic of 2,900,000 packets/day, had clearly reached a stage of operational stability, heavy usage, and was by any measure a major success. It was at this point that I left ARPA to spread the technology to the commerical world.

The ARPANET has continued to grow since 1973, with 111 host computers in 1977 and more than 400 hosts in all the interconnected networks by 1983. The network has become a utility for both ARPA and the Department of Defense as a whole. Research has continued into internetting and packet radio but little change has occurred in the basic network.

1969 Crossover

In 1974, after it was clear that the technology worked, I finally spent time analyzing the economic trends which made packet switching possible.(15) These trends could have been analyzed in the mid-sixties if someone were to have asked the right questions, but as it often turns out they were examined after the event they could have predicted. Simply, since packet switching requires more computation than circuit switching but saves transmission band- width, the trend analysis looked at the cost of computation and the cost of communications and found the situation illustrated in Fig. 3. For packet switching the cost of computation dominated in the early 1960's but in 1969 the cost curves crossed and afterward the cost of communications dominated. The composite cost of packet switching thus fell below the cost of circuit switching also about 1969 and since then the margin of advantage has quickly widened to where it equals the full peak to average transmission ratio for data of about 8 to 1. Subsequently, in a 1982 paper (16) I did a similar analysis for voice and concluded the crossover there was in 1978 and therefore packet voice would have been cheaper. In a somewhat similar fashion to data, the industry may recognize this ten to fifteen years after the event.

FIGURE 3



INCREMENTAL COST OF MOVING 1M BITS (1 KILOPACKET) ON A NATIONWIDE NETWORK

Impact on Computer Resources

One of the original goals of the ARPANET was resource sharing to optimize computer resources. In 1973 I surveyed the ARPANET users to see what alternative computer facilities they would need if they did not have the network. In a number of cases there were ARPA projects which did not have a computer at all but utilized computer power from resource centers which had developed. They obtained access through a TIP, typically onsite, and used one of a collection of computers often accross the country. They were generally happy not to have to run their own computer and had far more reliable service than if they had been limited to one machine. Also due to the statistics of large groups and the benefit of multiple time zones (from Hawaii to Europe) the number of computers required for these users was far less through consolidation. Another group of users were using the Illiac IV remotely for large numerical problems and would have required their own supercomputer or perhaps had to commute if the network were not there. In other cases individuals needed to use software unique to foriegn hardware and thus needed access to other computers besides their own. After finding rational solutions (fiscally sound) to all the ARPA users needs assuming no network, the computing cost was compared to the then actual computer cost and found to be **three times as expensive**. Independent of the details of the survey, it is clear that huge savings were possible (in that era of mainframes) by pooling computing demand. Today when all my computer resources come from micro-computers I couldn't save anything using remote computers for computing power. Supercomputer users still might save but in large part this benifit is mainly historical.

Other Network Development

CYCLADES:In France the interest in computer networks grew quickly during the early 1970's. In 1973 the first hosts were connected to a network called CYCLADES which linked several major computing centers throughout France. (17) The name CYCLADES refers to both the communications subnet and the host computers. The communications subnet by itself is called CIGALE and is a pure **datagram** network, moving only disconnected packets and delivering them in whatever order they arrive without any knoledge or concept of messages, connections or flow control. The ARPANET also operates using datagrams but perhaps the most avid supporter of the concept is the designer of CYCLADES, Louis Pouzin. As with any datagram network, a large part of the communications funtions must be implemented in the host computers; the packet ordering, message formation, flow control, and virtual connections support.

RCP: Another packet network experiment was started in France at about the same time by the French PTT Administration (18). This network, RCP, first became operational in 1974 and was the experimental testbed for the French public network service TRANSPAC. The design of RCP, directed by Remi Despes, differed sharply from that of CIGALE and ARPANET being organized around the concept of the **virtual circuits** rather than datagrams. RCP's character as a prototype public network may have been a strong factor in this difference since a vitrual circuit service is more directly marketable, not requiring substantial modifications to the customers' computer. In any case, RCP pioneered the incorporation of individually flow-controlled virtual circuits into packet switching networks.

Telenet: The success of the ARPANET led Bolt Beranek and Newman (the primary contractor for the ARPANET) to establish a commerical network company, Telenet in late 1972. In October 1973 I joined Telenet as president, and we filed with the FCC to become a carrier and construct a public packet switching network. The FCC approved the request six months later, and Telenet started the world's first public service in August 1975 with 7 nodes. By April 1978 Telenet's network had grown to 187 nodes providing service to 180 host computers and supporting direct terminal access service in 156 cities and interconnections to 14 other countries. Telenet was designed from the start to appear to the user as a **virtual circuit** service with the host interface being implemented over a communications line rather than with a box on site. However, for the first several years Telenet operated a core network based on datagrams copied from the ARPANET but implemented virtual circuits at all interfaces. It wasn't until the complete shift was made to Telenet's TP-4000 packet switch around 1980 that the savings of vitual circuits in the core net could be realized (about 30% for Telenet with a 32 byte average packet size).

EIN: Orginally known as COST Project II and later as the European Informatics Network, EIN is a multi-nation funded European computer network (19,20). The project director was Derek Barber of NPL, one of the original investigators of packet switching in the U.K. It was organized in 1971 but due to the difficulities of multi-national funding it did not become operational until 1976.

Other Public Data Networks: Public packet networks were announced in the U.K., France, Canada, and Japan in the 1973-1974 period. In 1977 the Experimental Packet Switched Service (**EPSS**) in the U.K. and **DATAPAC** in Canada became operational. Also in 1977 a second public network **TYMNET** became approved in the U.S. In 1978 **TRANSPAC** in France and **DX-2** in Japan became operational as public networks. By the early 1980's public packet network service was available in most countries in the world as an international service and as a domestic service in many of these countries.

CCITT Standards- X.25

With five, independent, public packet networks under construction in the 1974-1975 period (U.S.-Telenet, Canada, U.K., France, Japan), there was strong incentive for the

nations to agree on a standard user interface to the networks so that host computers wuld not have unique interfacing jobs in each country. Unlike most standards activities, where there is almost no incentive to compromise and argee, carriers in separate countries can only benefit from the adoption of a standard since it facilitates network interconnection and permits easier user attaachment. To this end the parties concerned undertook a major effort, to agree on the host-network interface during 1975. The result was an agreed protocol, CCITT Recommendation X.25, adopted in March 1976.

The X.25 protocol provides for the interleaving of data blocks for up to 4095 virtual circuits on a single full-duplex leased line interface to the network, including all procedures for call setup and disconnection. A significant feature of this interface, from the carriers' point of view, is the inclusion of independent flow control on each virtual circuit (VC); the flow control enables the network (and the user) to protect itself from congestion and overflow under all circumstances without having to slow down or stop more than one call at a time. In networks like the ARPANET and CYCLADES which do not have this capability, the network must depend on the host (or other networks in interconnect cases) to insure that no user submits more data to the network than the network can handle or deliver. The only defense the network has without individual VC flow control is to shut off the entire host (or internet) interface. This, of course, can be disastrous to the other users communicating with the offending host or network.

Another critical aspect of X.25 is that it defines interface standards for both the host-to-network block transfer and the control of individual VC's. In datagram networks the VC interface is situated in the host computer; there can be, therefore, no network standard for labeling, sequencing, and flow controlling VC's.

The March 1976 agreement on X.25 as the technique for public packet networks marked the begining of the second phase of packet switching: large interconnected public service networks. In the years since X.25 was adopted, many additional packet standards have been agreed on as well. X.28 was adopted as the standard asynchronous terminal interface and X.29, a protocol used with X.25 to specify the packetizing rules for the terminal handler, was adopted as the host control protocol. Also, a standard protocol for interconnecting international networks, X.75 has been adopted.

Packet Switched Voice

Packet networks today typically only utilize trunks and lines up to 64 kilobits/sec in speed. As a result, end to end delay in these networks remains high - over 100 ms. Network delay would however be vastly reduced given high speed trunks (T1-T3) and high speed packet switches (>100,000 packets/sec). Even with a shift to T1 trunks, the network delay would drop to 10 ms plus propagation. With T2 trunks, the delay would not be practically different than with a circuit switch (1-2 ms + propagation). For most data applications this is unnecessary but for voice it is critical. The voice network will often return an echo from analog phones and even 20 ms of delay can make such an echo sound bad. Short packets like 32 bytes (4.5 ms of 56 kb voice) are also necessary to reduce delay. But with high speed trunks and short packets, packetized voice is not only toll quality but also saves 67% of the transmission and switch bandwidth due to silence surpression. Thus in designing tomorrow's central office switch and toll switchs for the telecommunications network it is clearly extremely desirable to use packet switches for voice of any rate (56kb, 32kb, 16kb, etc), for data of all speeds and for switched video. Without packet switching the flexibility is vastly reduced and each service must be switched separately. Packet switches of the required speed are both feasiable and economic today if restricted to fixed-length, short packets (16 or 32 bytes), all flow contol and error checking are done at the network interfaces - not for each link, and a forshortened virtual circuit header is used (about 1 byte) with only the link number. AT&T has a prototype switch of 2 million packets/sec in test and designs exist for switchs up to 35 million packets/sec (21).

The most interesting facet of high speed switch technology is that for large central office or tandem office switches the peak switch throughput is limited by the speed of the fastest memory for a time switch and similarly for a packet switch but the packet switch is able to switch a whole packet not just one byte like the time switch. Thus in general for 32 byte packets, **packet switches can be made about 32 times the maximum throughput of circuit switches**. This becomes very valuable at high level tandem switching offices to switch traffic between many fiber paths.

Based on the importance of coping with variable bitrate demands for data, voice, and video and the cost savings for integrating all services with one switch it is clear that **the whole telecommunications plant will convert to packet switching in the future.** The date is harder to predict but conversion should occur during the 1990's and be complete by 2000. The basic service will be small packets on virtual circuits and each application will have its own interface units. Data services will use an X.25 interface which does the flow control and error checking for the VC on a end-to-end basis.

Future Computer Networks

If we look forward to the year 2000, based on historical trends computers will be about 500 times cheaper per computation than today. Almost all devices in the home, office or plant will have complex computers incorporated into them. Simple local area network facilities like shielded twisted pair will connect the majority of these together and a gateway will connect this local network to the public telecommunication system over

(hopefully a revised and truly integrated) ISDN facility. Distinctions as to which device is the phone may become difficult because many of the device-computers may use the voice network from time to time for warnings, ordering, and certainly in assisting us to find the party to whom we wish to talk. Phone service today is in great need of automation; we waste large amounts of time non-productively answering and calling to find people. Thus as the telecommunications network brings data and voice together home and office automation will integrate them.

REFERENCES

(1) P. Baran et al, "On Distributed Communications, Vols. I through XI," RAND Corporation Memos, August 1964.

(2) T. Marill and L. G. Roberts, "Toward a Cooperative Network of Time-Shared Computers," Proceedings FJCC 1966, pp. 425-431.

(3) L. G. Roberts, "Multiple Computer Networks and Intercomputer Communication," ACM Symposium on Operating System Principles, October 1967.

(4) D. W. Davies, K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," ACM Symposium on Operating System Principles, October 1967.

(5) L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proceedings SJCC 1970, pp. 543-549.

(6) F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," AFIPS Conference Proceedings 36, June 1970, pp. 551-567.

(7) L. Kleinrock, "Analytic and Simulation Methods in Computer Network Design," AFIPS Conference Proceedings May 1970.

(8) H. Frank, I. T. Frisch, W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," AFIPS Conference Proceedings, May 1970.

(9) S. Carr, S. Crocker, V. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," AFIPS Conference Proceedings, May 1970.

(10) S. M. Ornstein, F. E. Heart, W. R. Crowther, S. B. Russell, H. K. Rising, and A. Michel, "The Terminal IMP for the ARPA Computer Network," AFIPS Conference Proceedings 40, June 1972, pp. 243-254.

(11) L. Kleinrock, "Performance Models and Measurement of the ARPA Computer Network," Proceedings of the International Symposium on the Design and Application of Interactive Computer Systems, Grunel University, Uxbridge, England, May 1972.

(12) L. Kleinrock and W. Naylor, "On Measured Behavior of the ARPA Network," AFIPS Conference Proceedings, Vol. 43, NCC, Chicago, Illinois, May 1974, pp. 767-780.

(13) L. Kleinrock and H. Opderbeck, "Throughput in the ARPANET -- Protocols and Measurement," Fourth Data Communications Symposium, Quebec City, Canada, October 1975, pp. 6-1 to 6-11.

(14) H. Frank and W. Chou, "Topological Optimization of Computer Networks," Proceedings of the IEEE, Vol. 60, No. 11, November 1972, pp. 1385-1397.

(15) L. Roberts, "Data by the Packet," IEEE Spectrum, Vol. 11, pp. 46-51, February 1974.

(16) L. Roberts, "Packet Switching Economics," presented at L. M. Ericsson Prize Presentation, Stockholm, Sweden, May 1982.

(17) L. Pouzin, "Presentation and Major Design Aspects of the CYCLADES Network," Third Data Communications Symposium, Tampa, Florida, November 1973, pp. 80-85.

(18) R. F. Despres, "A Packet Network with Graceful Saturated Operation," Proceedings ICCC, Washington, D.C., October 1972, pp. 345-351.

(19) D. L. A. Barber, "The European Computer Network Project," Proceedings ICCC, Washington, D.C., October 1972, pp. 192-200.

(20) D. L. A. Barber, "A European Informatics Network: Achievement and Prospects," Proceedings ICC, Toronto, Canada, August 1976, pp. 44-50.

(21) A. Thomas, J. P. Coudreuse, M. Servel, "Asynchronous Time-Division Techniques: An Experimental Packet Network Intergrating Videocommunication," Proceedings International Switching Symposium, Florence, Italy, May 1984.

Mathematical Laboratories: A New Power for the Physical and Social Sciences

Glen J. Culler

The concept of a mathematical laboratory has been developing throughout the lifetime of computers. The capabilities made available in systems supporting these laboratories range from symbolic integration, differentiation, polynomial and power series manipulation; through mathematical simulation; to direct control experimental systems. About 1961 two trends, one toward what has become known as *on-line* computation, the other toward *time sharing* gained enough recognition to develop national support and subsequently have come to represent what is now known as modern computation. An on-line system provides interactive facilities by which a user can exert deterministic influence over the computation sequence; a time-sharing system provides a means by which partial computations on several different problems may be interleaved in time and share facilities according to predetermined sharing algorithms. For reasons of economy it is hard to put a single user in direct personal control (on-line, that is) of a large scale computer. It is equally or more difficult to get adequate computation power for significant scientific applications out of any small scale economical computer. Consequently, on-line computing has come to depend upon time-sharing as its justifiable mode of implementation. On the other hand, valuable on-line applications have formed one of the major reasons for pushing forward the development of time-sharing systems. At present, both efforts have reached such a stage of fruition that one finds many systems incorporating selective aspects of the early experimental systems of both types.

In this chapter, we will bring out some of the key features of highly interactive direct control systems that have implications for continuing design efforts aimed at furthering the development of experimental mathematical laboratories. We then present a brief description of the foundations of an existing facility at the University of California at Santa Barbara and illustrate its use in a typical application. Finally, we discuss some extension of the system which will provide deeper power for future experimental applications.

Reprinted from Interactive Systems for Experimental Applied Mathematics, Proceedings of an ACM Symposium. Edited by Melvin Klerer and Juris Reinfelds. © 1968, Association for Computing Machinery, Inc. Published by Academic Press, Inc.

1. The Operation-Operand Orientation for Direct Control

A primary aspect of modern computation in contrast to early computation is that of direct user control of computational processes. This is sometimes confused with very early "open shop" computing, in which the user was his own operator, but actually there is no similarity. The fact that a user in direct control commands his own console is only superficially analogous to his being his own operator: the important distinction lies in the program power for direct control associated with his console. This is relatively new and has required some software (computer programming) sophistication over and beyond that of "assemblers" and "compilers". At present, the logical structure required for direct control is well understood and can be simply described. Consider, first some examples of direct control with which we are all familiar; for example, a musician playing an instrument, a person driving an automobile, an operator running a crane, and a child on his skateboard (or surfboard, for that matter). In each instance, there is a lively dynamic situation in which the person is actively involved as the guiding force. A certain amount of prior training is required for success and the variance in training and skill over all people is pronounced. Also, in each case, thoughtful designers have attempted to improve the performance capabilities through the nature of the controls provided. But, no matter how well designed a piano may be, it is not possible for all people to play equally well. It may be the case that, the greater the control facilities, the larger the deviation in final performance. Thus, the essence of design quality should lie in the ultimate performance capabilities under talented control rather than in the elementary creature convenience for the initial learning period of the training. This is not to say that learning difficulties should be ignored, but that ultimate power should not be sacrificed for early convenience.

To design a direct control system, we first isolate the controlling elements (the operators) and their effects, (the transformation of the operands) and then provide a means for their interactive combination. For the design to be really successful, a deep understanding of the nature of the users and their performance desires is required. In the context of human involvement, the existence of Stradivarius violins and the yearly change of automotive design attest to the difficulty of the total direct control problem.

Abstracting from the examples above, we see that a logical structure for direct control consists of a set of operations together with a set of appropriate operands on which the user (a performer, driver, or athlete) can determine which operators

are to be applied to which operands during dynamic, real-time activity. If there, are no operators that make it possible to concatenate sequences of other operators, the system is said to be manual; when there are concatenation operators, the system is said to be programmable and a program is precisely a concatenation of operators with specified operands. If the operands must be absolutely specified at the time the concatenation is formed (i.e., when the program is constructed) then the system is absolutely programmable; if the operands may be specified by variables, then it is variably programmable; if the operators may be specified by variables, then it is substitutionally programmable.

As regards the selection of definitions of particular operators for a given application, very little can be generally said beyond the statement that the designer should try very hard to isolate the most maneuverable set of operators as primitives; this tends to shorten and simplify the programs obtained through concatenation. But there are some observations we can make pertinent to the selection of operators that are independent of the applications area.

RESET

> If there is a concatenation operator, then there should also be a reset operator which can override all other operators and force a stable halt, thus returning the system to manual control.

TEST

> There should be some from of logical operation that permits a conditional selection of a successor (in computer terminology this is called a brance instruction).

ENTER

> There should be some form of communication operator whereby the user can insert manual responses to program requests without destroying the requesting program's control.

REPEAT

> A fixed sequence of operations should be able to be recalled and performed any specified number of times.

Of course, we do not suggest that there is only one way of defining and utilizing the above facilities. Indeed, they can be very simple and minimal in capability or elaborate and extensive according to taste; but if the system does not have at least a minimal form of each of the above, then if suffers substantial restriction of its ultimate power.

Concerning concatenation operators, we have found it convenient to include several types within the same system. The reason for this is that with distinguished concatenations we are able to provide embeddings of program control that enrich the logical structure of the system. We'll return to illustrate this point later when we discuss console programming.

As a means to give specific explanation of how direct control systems can be used in mathematical laboratories, we now formulate an operational calculus in the area of discrete analysis and then detail its implementation in a direct control system. Following this, we will show its application in a laboratory system for the study of speech wave forms and finish the chapter with a brief description of the extension to more complicated data structures and associated operations.

2. An Operational Calculus for Discrete Analysis

The set of operations defined below have been chosen for their simplicity and ease of combination and have with-stood the test of experimental usage for several years and in several systems. The operands include single numbers, lists of numbers (one-dimensional arrays), double lists of number (two-dimensional arrays) and matrices; all of these are allowed to be either real or complex. Since the operands are of several categories, we must include a means for specifying the category. One way to do this is to set aside special variables for each category as illustrated by the expression:

$$f(x) = \sum_{j=0}^{\infty} \left(\sum_{k=1}^{j} A_{jk}\, e^{-kx} \right) \frac{x^j}{j!}$$

which shows the almost unwritten but broadly accepted rules of mathematical notation. THis may well be the best approach when the range of categories is properly restricted and when the conventions of a particular problem area are adhered to. But when many application areas utilize the same system each with their own notational habits, then this approach becomes unwieldy, though certainly not impossible. We select another approach which has been quite successful in practice. This consists of explicitly declaring the category; that is, of providing category operators:

REAL Operands shall be real.

CPLX Operands shall be complex.

I Operands denote single numbwers.

II Operands denote one-dimensional lists of numbers.

III Operands denote two-dimensional lists of numbers.

IV Operands denote matrices.

The operators are defined in relation to the operands as detailed below; they can be grouped as local and global by the nature of their definitions; local operators can be further grouped as arithmetic or as elementary functions. The main content of the definitions is best illustrated for II REAL; that is, for operators on one-dimensional real arrays. Thus in the definitions below, let each letter A,...Z denote a list of n components

$$A = (a_1, a_2, a_3, ..., a_n)$$

such that each component [figure] n is substituted by a real number prior to any actual computation.

A. Operator Definitions for II REAL

1. Local operators are those which operate local to a coordinate index. This means increasing or decreasing the number of coordinates would not essentially change the consequence of these operations at earlier coordinate positions.

a. The arithmetic operators consist of $\oplus$, $\ominus$, $\odot$, $\oslash$, SQ, SQRT, NEG, INV, DIFF, SUM, MOD:

$X \oplus Y = (x_k + y_k \mid k = 1, \ldots, n)$

$X \ominus Y = (x_k - y_k \mid k = 1, \ldots, n)$

$X \odot Y = (x_k \cdot y_k \mid k = 1, \ldots, n)$

$X \oslash Y = (x_k / y_k$, if $y_k \neq 0$; 0, if $y_k = 0 \mid k = 1, \ldots, n)$

$SQ\ Y = (y_k^2 \mid k = 1, \ldots, n)$

$SQRT\ Y = (\sqrt{y_k}$, if $y_k > 0$; 0, if $y_k \leq 0 \mid k = 1, \ldots, n)$

$NEG\ Y = (-y_k \mid k = 1, \ldots, n)$

$INV\ Y = (1 \oslash Y)$

$DIFF\ Y = (y_{k+1} - y_k \mid k = 1, \ldots, n-1;\ 2y_n - 3y_{n-1} + y_{n-2} \mid k = n$

$SUM\ Y = (\sum_{j=1}^{k} y_j \mid k = 1, \ldots, n)$

$MOD\ Y = (|y_k| \mid k = 1, \ldots, n)$

b. The elementary function operators consist of SIN, COS, LOG, EXP, ATAN, ARG, DEL.

$SIN\ Y = (\sin y_k \mid k = 1, \ldots, n)$

$COS\ Y = (\cos y_k \mid k = 1, \ldots, n)$

$LOG\ Y = (\ln y_k$, if $y_k > 0$; 0, if $y_k \leq 0 \mid k = 1, \ldots, n)$

$EXP\ Y = (e^{y_k} \mid k = 1, \ldots, n)$

$ATAN\ Y = (\tan^{-1} y_k \mid k = 1, \ldots, n)$

$ARG\ Y = (\pi$, if $y_k < 0$; 0, if $y_k \geq 0 \mid k = 1, \ldots, n)$

$DEL\ Y = (1$, if $y_k = 0$ or if $y_k y_{k+1} < 0$ and $|y_k| \leq |y_{k+1}|$ or if $y_k y_{k-1} < 0$ and $|y_k| < |y_{k-1}|$; 0, if otherwise $\mid k = 1, \ldots, n)$

2. Global operators are those which do not operate local to coordinate indices and are those for which increasing or decreasing the number of coordinates would effectively alter the consequences of the operation at earlier coordinate positions. These are LS, RS, REFL, MAX, ID, EVAL;

$$\text{LS } Y = (y_2, y_3, \ldots, y_n, y_1)$$

$$\text{RS } Y = (y_n, y_1, y_2, y_3, \ldots, y_{n-1})$$

$$\text{REFL } Y = (y_n, y_{n-1}, \ldots, y_2, y_1)$$

$$\text{MAX } Y = \left(\max_{j = 1, \ldots, n} y_j \;\middle|\; k = 1, \ldots, n \right)$$

$$\text{ID } Y = \left(\frac{2k - n - 1}{n-1} \;\middle|\; k = 1, 2, \ldots, n \right)$$

$$(X, Y) \text{ EVAL } U = (U, Y(U)) \text{ to first order.}$$

This later operation requires further specification. First extend X by augmenting with $x_0 = -\infty$ and $x_{n+1} = +\infty$

$$-\infty \le t \le \min_{k=1,\ldots,n} X_k \quad Y(t) = Y(x_{min})$$

$$\max_{k=1,\ldots,n} X_k \le t \le \infty \quad ; \quad Y(t) = Y(x_{max})$$

Finally, for each value of U, (i.e., u_j, j=1, ...,m; m not necessarily equal to n) determine x_μ and x_ν such that

$$x_\mu = \left(\max_{k= 0, \ldots,n+1} \;\middle|\; x_k \le u_j \right)$$

$$x_\nu = \left(\min_{k= 0, \ldots,n+1} \;\middle|\; x_k > u_j \right)$$

and choose

$$y_j = y_\mu + \frac{u_j - x_\mu}{x_\nu - x_\mu} (y_\nu - y_\mu).$$

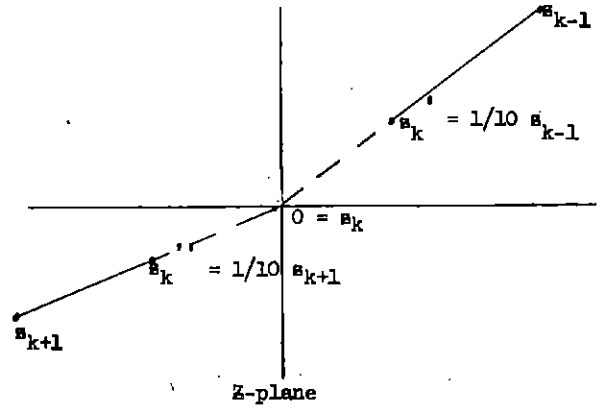
In the exceptional case when some of the u_j's fall outside the range of X, the extension of Y given above gives Y one constant value to the left of x_{min} and another constant value to the right of x_{max}. Also, notice that no ordering of x-values or the 8-values has been assumed; this is important for the required generality of the EVAL operator. As defined, it provides a basis for graphical composition of functions and makes it possible to change discrete variables in the middle or a calculation. When some of the functions occurring in a computation are only known discretely and not at the same intervals used in the calculation, then some form of this EVAL operator is mandatory.

B. Extension of II CPLX

The extension of II REAL operators defined above to the II CPLX operators is accomplished by first introducing complex lists:

$$Z = X + iY; \qquad z_k = x_k + iy_k, \; k = 1, \ldots, n$$

and then defining the arithmetic operators using complex arithmetic on corresponding coordinates. All this is immediate up to the selection of branch cuts as required in the definition of LOG and SQRT. The best way we have found for maintaining compositional compatibility is to define ARG with the facility of branch specification and define all other functions requiring branch definitions in terms of ARG. If we think of a one-dimensional complex array as an ordered set of vertices of a polygonal arc in the complex plane, then the change in argument at adjacent vertices is not more than 2π, consequently, we can computer this change using either the positive or negative principal branch of the argument function. In the event that one of our vertices (Φ_k) actually coincides with the origin, we augment that array and replace that vertex with two new vertices (Φ_k', Φ_k'') chosen on the adjacent polygonal segments..



The final argument function, ARG_+ or ARG_- is just the II REAL SUM of the argument difference. Of course, both ARG and MOD of II CPLX are real valued functions, thus part of their calculation consists of making their imaginary parts equal to zero. Also, because conjugation plays such an important role in complex structures, we choose to drop the NEG operator and replace it with *, complex conjugation. Since the real operators when composed turn out to be transformations on Y-coordinates, both NEG and * have the same result: the change in sign of the Y-coordinates.

The global operators also extend easily; LS and RS are direct, REFL brings in a new facility

Let $Z = X + iY \quad X, Y, \text{II REAL}$

$$\text{REFL } Z = \text{REFL } (X + iY) = Y + iX,$$

thus exchanging real and imaginary parts. Coupled with REAL REFL we can still quickly reverse the orderig of a complex array

The extension of MAX results in:

$$\text{MAX } (X + iY) = \text{MAX } X + i \text{ MAX } Y \qquad X,Y \text{ II REAL}$$

The extension of EVAL to II CPLX is not well defined; for it to be so requires strenuous conditions on the arc at which the initial pair is being evaluated, and the operation does not appear to be computationally valuable. The situation is much improved, however, for EVAL III CPLX, which we will discuss later. The operator ID is a fundamental array generator; for II REAL, it produces an interval from -1 to +1 with n equally spaced values; for II CPLX it produces selectively a discrete representation of the unit circle (i.e., the nth roots of unity) or a square about the origin joining (-1-i, 1-i, 1+i, -1+i) with equally spaced points.

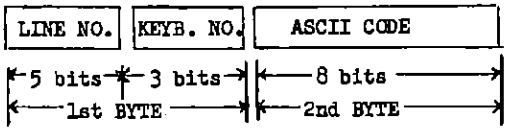
C. Contraction to I REAL and CPLX

Even in problems that are primarily array oriented, we frequently need to use and produce scalars, both real and complex. Consequently, part of our operational calculus consists of operators of category I. The definitions of these operators are obtained by setting n = 1 for category II operators and discarding those that lose their meaning; as for example, SUM, DIFF, MAX, LS, RS, REFL, EVAL, etc.

Starting from this bare frame of operators, we must now show how to accomplish significant control of a computer system and obtain the facilities of a mathematical laboratory.

3. Construction of a Mathematical Laboratory

To build a system that under direct control performs the operations just defined we first introduce a keyboard oriented especially to those operations. Each time one of the keys is depressed a number uniquely corresponding to that key is transmitted to our computer and stored in its memory. The program within the computer analyses this number and determines the type of key just depressed, then responds in accordance with conditions already set up by the antecedents of this key. We make this explicit for our operational calculus. First, observe the keyboard in Figure 1. It is divided into an upper-half and a lower-half; we designate the upper-half as operators and the lower-half as operands. The number transmitted by depressing a key has two parts, the first is the name of the keyboard and the second is the number of the operator key or operand key. These are, of course, binary numbers and are chosen to agree with the American Standard Code for Information Interchange. The format of the number stored in the computer is:



where the line number refers to the current incoming telephone connection is inserted by the equipment servicing the telephone lines, the keyboard number and the ASCII code for the depressed key actually come in on the telephone line. If no telephone lines are used, that is if the keyboard is physically near the computer, then the line number is taken to be zero.

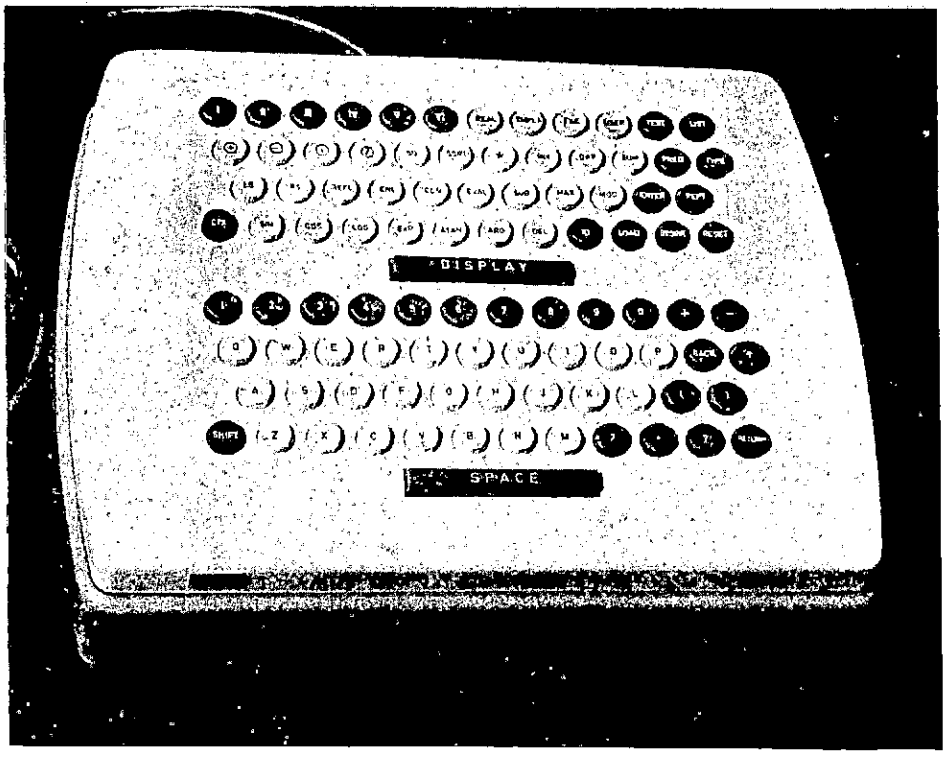


Figure 1. An operational calculus keyboard for discrete analysis.

The operators are grouped according to their system properties and typical use:

A. Operator Category Keys - Blue I, II, III, IV, V, VI.

B. System Change Keys - Yellow REAL, CPLX, SYST, USER.

C. Program Control Keys - Green, Black, Red LIST, TEST, PRED, TYPE, ENTER, REPEAT, DISPLAY, ERASE, LOAD, STORE, RESET.

D. Application Operator Keys - White + - * / SQ SQRT *NEG INV DIFF SUM LS RS REFL ID EVAL MAX MOD SIN COS LOG EXP ATAN ARG DEL.

The operand keys are also grouped according to their usage:

A. Numerical Keys - Blue

B. Alphabetic Keys - White

C. Punctuation Keys - Green

D. Typing Control Keys - Black

Using such a keyboard we can direct the activities of our computer by the sequence of keys depressed. But to do this well, we need to have an accurate mental image of what the computer does in response to each sequence. From the point of view of communication, this is analogous to reading or doing mental arithmetic or in feeling which note should be played next on a musical instrument. The issues involved in bringing this about are threefold:

1. Understanding when an operator-operand sequence is informationally complete,

2. Knowing the family of operand specifications for each operator used,

3. Keeping track mentally of the consequences of each transformation performed.

It is clear that not everybody will do this equally well, but those who use direct control systems with power soon develop these capabilities. A different approach, which is analogous to pushing a bicycle instead of riding it, is to use interactive cues from the computer which force control to go along certain preprogrammed sequences. This is especially useful during the initial training period, but should be curtailed when the user's facility has developed. Systems with too much cueing soon become loggish and stultifying to the able user.

Our method for making the required mental images easy, is to associate a so-called working register with each operator category I, II, III, IV, etc. and picture the consequence of each informationally complete operation as being placed in this working register. As one changes operator categories, the contents of the associated working registers are not changed and hence upon returning to a given category, its contents remain as previously constructed. It is not necessary to prefix each operator with its category designation, that is, the operator category is maintained until a new category is specified. To be more specific about this, we discuss some of the operators in relation to their implied transformations of the working registers. Let the working register be denoted by [figure] and its two associated sub-registers by α and β. To distinguish register categories we use a category subscript:

$$\zeta_I = (\alpha_I, \beta_I)$$

$$\zeta_{II} = (\alpha_{II}, \beta_{II}), \text{ etc.}$$

Now on II REAL, a composition of functions like...

$$f(x) = \log \sin (3x^2 + 1)$$

represented correctly by ...

$$(X, \log \sin (3x^2 + 1))$$

where X is the independent variable running over its prescribed domain. To make such a set of ordered pairs, we construct the discrete domain for X

$$X = (x_1, x_2, ..., x_n)$$

and put in α_{II}, then compose the three transformations and put the result in β_{II}. We must be able to transfer data lists to and from both associated registers α and β of all categories; this is one of the uses of the LOAD and STORE operators.

LOAD:

> Move the data selected by the successor into the working register.

STORE:

> Move the data from the working register into the location defined by the successor.

A typical manner of operation is to load some initial data into B, then to transform it by a sequence of transformations and finally to save it away by storing it as some dependent variable. The point of having LOAD and STORE directly available is to provide specific control over the contents of the working register. On the other hand, there are many instances where the transfers to and from the working registers can be performed implicitly and at a saving of manual labor. Thus, we make implicit transfers also possible where convenient. For II REAL, the operations defined in Section 2 all leave the consequences in β_{II}, which satisfies the compositional conditions noted above. Each operation occurs in a number of situations according to is predecessors and successors, but within the operator sub-groups defined in Section 2A, above, there is sufficient similarity that is adequate for us to illustrate a typical member of each sub-group.

Computations are carried out in complete compositional sequences, that statement of completion being any category key, DISPLAY, STORE, or RETURN. After a computation is performed, the results are left in β, whether or not they have been saved by the calculation. A new compositional sequence can be initiated by an operator or operand; if initiated by a variable operand which is not immediately followed by =, then an implicit LOAD is effected; if the variable operand is immediately followed by =, then the composition sequence will end by effecting an implicit STORE associated with that variable.

To illustrate the above relations of compositional sequences, we analyses equivalent sequences of computing F = log sin $(3x^2 + 1)$. Assume that we are using II REAL and that X has previously been constructed.

A. LOAD X SQ * 3 + 1 SIN LOG STORE F

B. $3X^2 + 1$ SIN LOG = F

C. LOG SIN $(3X^2 + 1)$ = F

D. F = LOG SIN $(3X^2 + 1)$

These all make the results of computing:

$$F_k = \text{LOG SIN } (3X^2 + 1)$$

available as the discrete data for the variable F, but they show different aspects of the direct control system. In a, the composition sequence is constructively ordered but greatly different in appearance from the original expression. After each step of A the contents of β_{II} are explicitly known to the user:

LOAD X	$(x_1, \ldots, x_n) \in \beta_{II}$
SQ	$(x_1^2, \ldots, x_n^2) \in \beta_{II}$
$\odot$ 3	$(3x_1^2, \ldots, 3x_n^2) \in \beta_{II}$
$\oplus$ 1	$(3x_1^2 + 1, \ldots, 3x_n^2 + 1) \in \beta_{II}$
SIN	$(\sin(3x_1^2 + 1), \ldots, \sin(3x_n^2 + 1)) \in \beta_{II}$
LOG	$(\log \sin(3x_1^2 + 1), \ldots, \log \sin(3x_n^2 + 1)) \in \beta_{II}$
STORE F	$(\log \sin(3x_1^2 + 1), \ldots, \log \sin(3x_n^2 + 1)) \in \beta_{II}$

In B, we begin with an implicit LOAD, use the power operator ↑ which does not appear except in forming the superscript, and replace STORE with which is equivalent in this context. In C, we use an interpretive ordering of the operators and obtain a familiar appearance as a formula but lose any apparent correspondence to the transformations performed. In D, we precisely have mathematical notation and are satisfied to be sure that F will end up with the right answers. In such simple circumstances it, of course, makes no real difference which approach is used. In practice, it depends upon the experimental requirements of the situation; that is, if there are no possible computational difficulties and all the compositional sub-sequences are unimportant to the user, then the automatic formula approach of D is most satisfying; but if something has gone sour, (say through singularities, bad distribution of data, or loss of accuracy through numerical pathologies) then it is important to be able to construct the results through step by step transformations of β_{II} and A is the most satisfying. Since both circumstances occur frequently in using a mathematical laboratory, we must provide both facilities (B and C are irrelevant, they are simple variations of A and D).

The final ingredient of the laboratory, is a display tube which permits visual understanding of the operations carried out by use of the DISPLAY operator and will be illustrated in the next section.



Figure 2. User consoles in the University of California, Santa Barbara mathematical laboratory.

4. Experimental Analysis of Speech Wave Forms

In a total system like that outlined above, there are so many operations each with several instances and special uses that we cannot present all contingencies and facilities short of producing a fairly large user's manual. But we can convey some sense of a user's experience to the reader by tracing through and explaining a brief but fairly typical actual sequence of operations as they occurred when we were studying how to analyze wave forms arising in natural speech. The equipment used consisted of an early mathematical laboratory built around the RW-400 polymorphic computer with the addition of an IBM 1800 process control computer. To the latter computer we attached a microphone input and an audio speaker output as well as a two-way data transfer connection to the first computer. Following the operator-operand philosophy outlined in Section 2, we selected a set of primitives that gave us control of the microphone input and the speaker output and programmed them for the IBM 1800. The control of these operations was implemented as a special category, VII, and added to the former mathematical system. The operations consist of controls for recording levels, for selecting which sub-lists of data to listen to, for selecting and transferring data to and from a back-up disk. Thus, we can digitally record live speech, bring it into the mathematically oriented system and decompose it by whatever means occur to us. In the other direction, we can manufacture mathematical functions and use them to build data lists in the IBM 1800; then make the 1800 output this list through an audio speaker for us to hear.

The data buffer, αVII, consists of 7440 memory locations subdivided into 60 blocks of 124 numbers each. We chose to use some letters on the alpha-numeric keyboard followed by specification of blocks and position within the blocks as non-numerical operations.

A. Category VII Operators

RETURN:

Initiate operation just specified.

R k:

Begin recording input data when the signal exceeds the threshold k (30 threshold levels are available).

Q:

Quiet, cease output to the speaker.

L a, b, c:

Listen; initiate output to the speaker starting with block a, position c, and output b consecutive blocks before delay.

D a:

After each speaker output delay a times a unit interval before repeating the output.

I a,,c:

Input 1 block starting at block a, position c and place the data in β_{II} of the RW-400 system.

O a,,c:

Output 1 block from β_{II} of the RW-400 system and put it in block a starting at location c.

S k:

Apply a k-point smoothing formula to that part of the data list in α_{VII} addressed by the last L a,b,c operator.

LOAD k:

Transfer the data on disk cylinder K into α_{VII} of the 1800.

STORE k:

Transfer the data list in α_{VII} of the 1800.

LS:

Specify a variable to be used as a vector of scales for the blocks of α_{VII}.

RS:

Rescale all blocks to a common scale.

B. Experimental Analysis of SAT

The pictures below showing a grid were taken with a Polaroid camera attached to a Tektronix storage oscilloscope that records the analog voltage signal output from the IBM 1800. The others not showing a grid were taken with the same camera attached to a second such oscilloscope used as a display device for alpha-numeric and graphical output from the RW-400.

To initiate a run, the user presses

RESET LOAD USER

and receives the computer's response requesting his user number. After the number is provided, the computer retrieves his complex of user programs and accumulated data and types out the user's own system message.

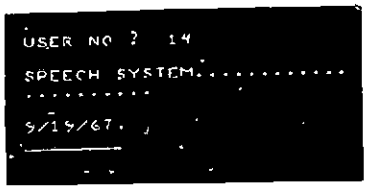


Figure 3. Start of a run

To record a word for study, we pressed

VII R 7 RETURN

and pronounced "sat" into the microphone. After adjusting the oscilloscope scales as shown below the graph, we see the complete wave form of the recorded word.

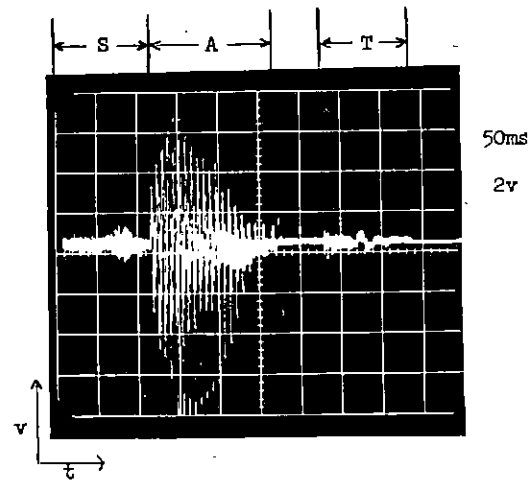


Figure 4. Wave form of SAT

In order to view the character of the wave form producing S we pressed

L 1, 17 RETURN

and adjusted as is shown. Since the oscillations were too rapid to see we then pressed

L 9, 1 RETURN

which allowed us to see block 9 alone, where the large amplitude of S occurred. Compressing the time scale resulted, finally, in a good picture of the mixture of wave functions in S.

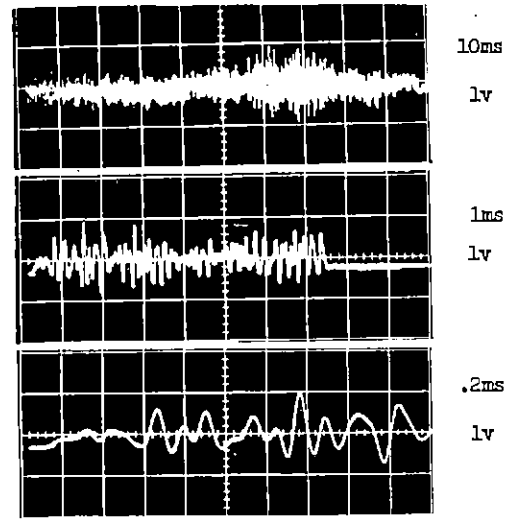


Figure 5. Details of the S wave form

Carefully observing the form of the last graph presented above allows one to identify the actual ingredients of S as a sequence of Gaussian wave functions, i.e. normalized derivatives of

$$EXP((-t^2/2\sigma^2)$$

having an assortment of k-values (the order of the derivatives) and occurring haphazardly in time. This decomposition of speech wave forms into Gaussian wave function components is a new approach to the classification, analysis and synthesis

of speech elements and, of course, cannot be properly treated here. But we can illustrate a simple instance of it. For this purpose, we will study the data producing the A in SAT.

The three pictures of Figure 6 were obtained respectively by

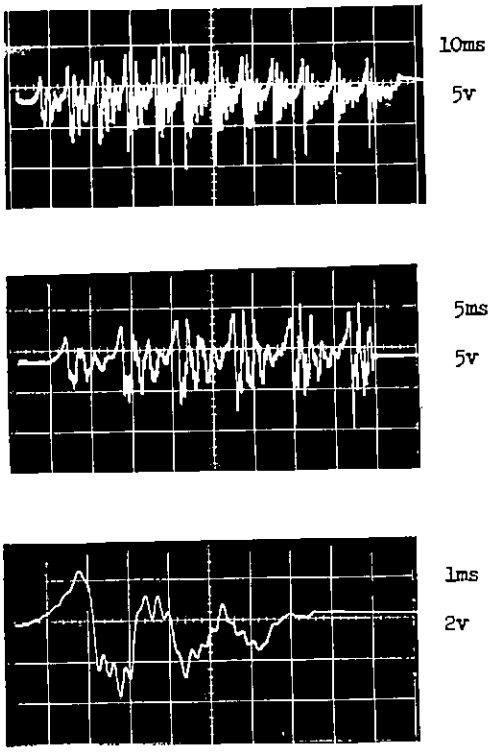


Figure 6. Pitch periods of the A in SAT

L 14, 14 RETURN

L 14, 6 RETURN

L14, 1, 70 RETURN

and show a successive blow-up of the initial pitch period of the A (the wave form lying between the larger extremes in a nearly recurrent manner is called a "pitch period"). From the first of these graphs, we can observe a relative rise in the second and third positive extremes. The fourth pitch period seems to be fairly representative of this family, so we all it into the mathematical system by pressing

I 18,,50

still of category VII, then enter category II REAL for the ensuing analysis, as shown in Figure 7, we then put the curve on maximum scale by adding zero,

\+ 0

and then save it as the variable operand Q and finally display the curve. Each such pitch period is made up of essentially three signals distinguished by their oscillations frequencies. There are many ways to attempt to separate out these sub-signals; we shall use the DIFF operator in combination with an integral mean smoothing operator as in VII S.K above, or a five point smoothing operator as shown in Figure 8. The graph in Figure 8 is the result of applying two successive smoothings. Subtracting this smooth function gives a first approximation to the high format.

Due to inherent properties of smoothing operators, we cannot be sure that this approximate formant is pure; that is, it may still contain some components of the two low formants. Running it through the same smoothing process results in a function of small amplitude.

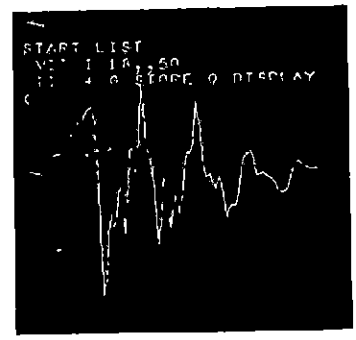


Figure 7. The fourth pitch period of A in SAT

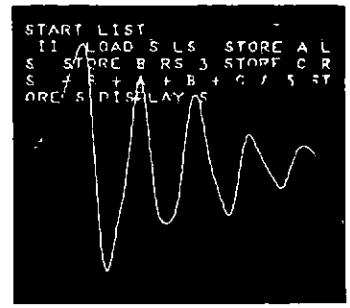


Figure 8. Combination of Low Formants

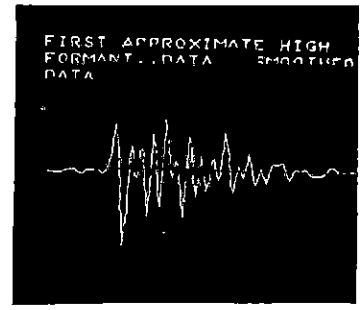


Figure 9. Approximate High Formant

When this function is subtracted from that shown in Figure 9, we obtain the extracted high formant as shown in Figure 10.

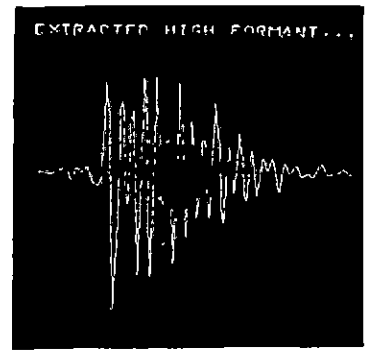


Figure 10. Second approximation to the high formant

Another method based on the DIFF operator uses the fact that for well separated frequencies, and fairly close data points, the difference operator separates the relative amplitudes of the formant. Applying DIFF DIFF NEG to the original data curve of Figure 7, produces the graph shown in Figure 11. The normalization of this function is lost in the DIFF process, so only shapes should be compared.

Returning to the consideration of the low formants, we can use DIFF again to try to split apart this pair. The dotted graph in FIgure 12 is the smoothed graph above, and the solid graph is its DIFF. Notice particularly the disfiguration of the solid graph near the large positive maximum, this shows where two wave functions are overlapping.

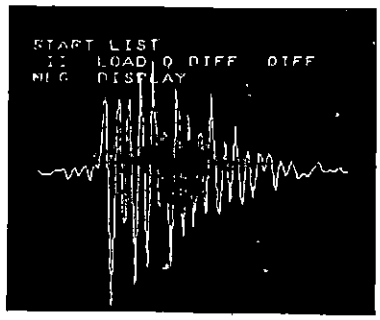


Figure 11. Unnormalized approximate high formant using the DIFF DIFF operator

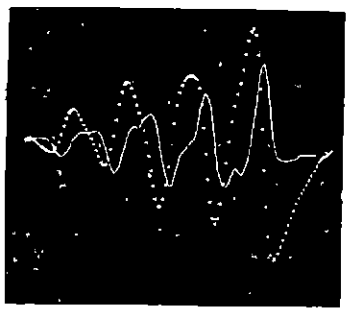


Figure 12. DIFF applied to the combined low formants

Our approach now is to identify the wave functions in this difference graph and compose them to approximate it, then to use the SUM operator to obtain an approximation of the dotted graph of Figure 12. Experimenting with the Gaussians width results in the choice of

$$1/2\sigma^2 = 81$$

as one which looks hopeful. To see this compare figure 13 with the shapes of the extremes in Figure 12. Our conjecture is that for this A sound, the deep first minimum results from the second derivative of the Gaussian and the second wave function occurring later is just its normalized derivative. Using the LS and RS operators we position the minimum and with NEG, MAX, INV we choose amplitudes and thus construct the wave functions shown in dotted curves of the first two graphs of Figure 14. Adding the resulting wave functions together, we next apply DIFF and then reposition and renormalize the result to fit the second pair of extremes as shown in the dotted graph of the third picture. Continuing this process once more then completes our approximation and using SUM results in the dotted graph of Figure 15, which is compared to the solid curve resulting from the smoothing operator. Now our approximation has no components of the lowest formant in it which accounts for the changes in elevation on the right hand side of Figure 15. Finally, adding in the high formants allows comparison of the original data curve with our fit.

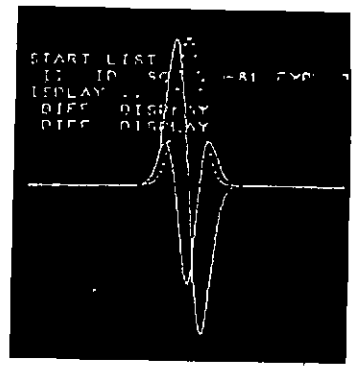


Figure 13. The first three Gaussian wave functions

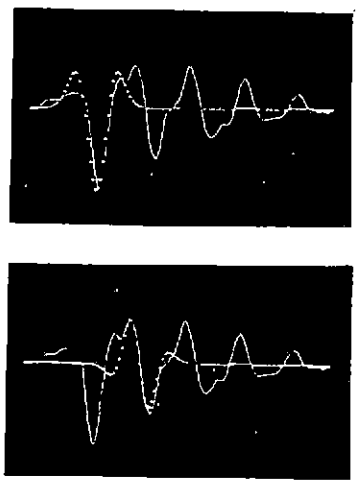


Figure 14. Fitting the difference curve by wave functions

If one looks carefully back at the extracted high formant, he will see one of its constituent wave functions. The procedure just outlined will provide a representation in the same manner and result in a total wave function representation of the original data curve.

Going back to the speech computer, we use

VII L 40, 19 RETURN

and get a good view of the wave making the T sound. Since we are always able to see and hear the data simultaneously, the identification of the linguistically effective parts of the wave forms is easily made. The data for T is similar to that

for S except for the single large slow wave function clearly evident in the T.

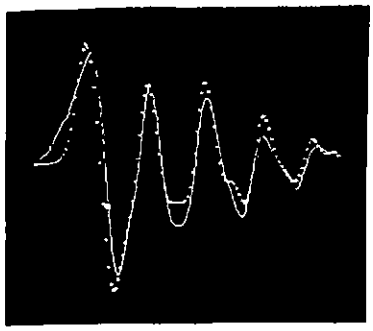


Figure 15. Comparison of smoothed data and wave function fit

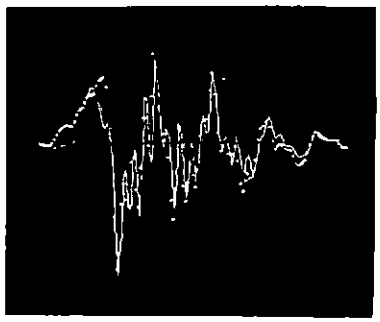


Figure 16. Comparison of original data and wave function fit

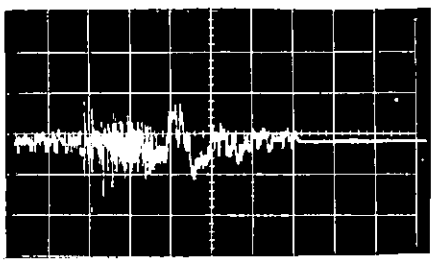


Figure 17. The wave function for the T in SAT

5. Extension or Higher Category Operations

The requirements of a structure supporting operations of categories II and IV both REAL and CPLX can be made clear by formulating several problems that point the way towards generalized data objects and their associated transformations.

A. Problems for Category III REAL

1. Integral equation

Let s, t be independent variables running over an interval [a,b] and F (s,t,u) a function of three variables such that for each point (s,t) ε $[a,b]^2$. F is a rational function of U. For some class of non-zero driving functions g(t) we wish to determine U(t) such that

$$U(t) = g(t) + \int_a^b F(s,t,U(s))\ ds.$$

To check whether a given function U(t) satisfies this equation the coefficient functions $A_j(s,t)$, $B_j(s,t)$ must be computed and the rational function of order J constructed.

$$F(s,t,U(s)) = \frac{\sum_{j=0}^{J} A_j(s,t)\ U^j(s)}{\sum_{j=0}^{J} B_j(s,t)\ U^j(s)}$$

If we choose III REAL precisely as a two-dimensional extension of II REAL, and use discrete variables

$$S = (s_1, s_2, \ldots s_n)$$

$$T = (t_1, t_2, \ldots t_n)$$

then the data structure for the coefficient functions

$$A_j(s,t) = (A_j(S_k,t_l) \mid k = 1, \ldots, n;\ l = 1, \ldots, n)$$

are the directly addressable operands for III REAL. The extension of the II REAL operators is achieved by considering the columns of III REAL operands (or rows) to be of the same class (but differently labelled) as II REAL operands. Then II REAL may be used sequentially to build III REAL operands.

2. Properties of surfaces over plane domain

Let H(x,y) be defined for points (x,y) on the closure of a domain D which consists of the interior of a simple closed curve Γ

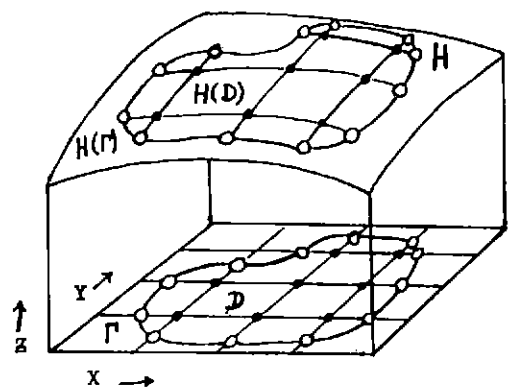


Figure 18. Surface representation of III REAL operands

Since D is not necessarily rectangular we must enrich our data representation by some form of logical selection which permits partial columns and partial rows to be easily used. Now let

$$X = (x_1, \ldots, x_n)$$

$$Y = (y_1, \ldots, y_n)$$

be two REAL operands. An operator W which forms the product set X x Y builds a discrete rectangular domain which is a natural operand for III CPLX, since corresponding to each index pair (j,k) there is a coordinate pair (x_j, y_k) which can be thought of as a

point on a plane grid. Observing that Γ can be represented as a II CPLX operand, we let

X x Y = W, III CMPLX

and

Γ = G, II CMPLX

then on III REAL we generate a real array of 1's and 0's which is the characteristic function of the interior of G with respect to W:

LOAD III CPLX W DEL II CPLX G

where G is insured to be closed by considering the first and last points to be adjacent vertices in a closed polygon. This characteristic function can be used as a logical mask in calculations involving D; also, it is binary by nature, so it can be compressed to bit by bit representation and carried along with W above as the representation of D. As shown in Figure 18 above, it is also convenient to interpolate Γ at the grid line crossings of the W array. This is accomplished on II CPLX by:

LOAD G EVAL III CPLX W

and results in a new representation of Γ consisting precisely of the interpolated intersections of the grid lines of W and the polygon Γ.

From the above, it should be remarked that generalizing DIFF selectively to partial differences and SUM selectively to row sums and column sums, and using ↑ as an "up" shift and | as a "down" shift to add to LS and RS already available, we have appropriate operand and operator representations to work with discrete methods for partial differential equations in two variables; these comments hold both for Marching problems and Relaxation problems. Furthermore, transformations of data representations are easily effected. For example, let:

$Z = (\zeta_1, \zeta_2, \ldots \zeta_l) \in$ II REAL

and

$H = (h_{jk} \mid j=1, \ldots, n;\ k=1, \ldots, m)$

then on III CPLX,

LOAD III REAL H EVAL II REAL Z

results in a complex array consisting of l each arc being a contour of Z = H(x,y) discretely approximated. That is, the above composition of operators constructs a contour map of H at elevations defined by the component of Z.

B. *Problems for III CPLX*

In many respects, the generalization of II CPLX to III CPLX is simpler and more apparent than the extension of II REAL to III REAL. The reason for this rests in the fact that II CPLX already involved plane considerations, and extensions to III CPLX essentially iterates the definitions if II CPLX. Thus, for example, let F be an analytic function over a domain D with boundary Γ. Let $\zeta_1, \zeta_2, \ldots \zeta_m$ be polygonal arcs in D. Choose the III CPLX operand

$Z = (\zeta_1, \zeta_2, \ldots \zeta_l)$

then $F(\zeta) = F(F(\zeta_1), F(\zeta_2), \ldots, F(\zeta_l))$

is the result of applying the "operation" F to Z and is an image arc representation. If D is well spanned by Z, that is, if each point of D is sufficiently near some points of Z, then F(D) is well spanned by $F(\zeta)$ because F was presumed to be analytic on the closure of D. If F is univalent over D, then $F(\zeta)$ is a discreet representation of the conformal mapping defined by F.

As mentioned in Section 2, III CPLX EVAL achieves an appropriate meaning. For example, suppose

$V = (v_1, v_2, \ldots, v_j) \in$ III CPLX

and that each v_j, j = 1, ... l lies in D where D and F are discussed above. The F(v) can be computed by an interpolation on III CPLX:

(Z, $F(\zeta)$) EVAL V

in which each point v_{jk} is located in Z by finding its nearest neighbors, and $F(v_{jk})$ is computed from the function values corresponding to these neighbors.

C. *Considerations for Category IV REAL and CPLX*

There are, of course, a variety of computational mathematical problems that do not fit naturally into an operator-operand format. For these, a well designed mathematical laboratory should be able to exercise library facilities and incorporate any advantageous programs, methods, etc. that exist in conventional systems into a user's own system. There is no reason to try to force an unnatural format on a given problem area. Just as this is "bad" mathematics, so is it "bad" computation. Consequently, it is important to recognize the constructive potentials and natures of mathematical endeavors and select computational approaches in resonance with these.

With this view in mind, we choose IV REAL to have operands that are symmetric matrices and represent these matrices in diagonal form. That is, our matrices are of the form:

M = U * D U

where U is a unitary matrix whose columns are eigenvectors of M and D is the diagonal matrix of eigenvalues of M. The data structure D is like a II REAL operand, and the data structure of U is like a III REAL operand. Now the elementary functions on M obey:

F(M) = U * F(D) U

and hence are computable by II REAL operations. Thus, our operational calculus extends readily to the algebra generated by M.

For IV CPLX, Hermitian matrices follow the above in detail and require no further discussion.

For general matrices, those that are diagonalizable still

permit reasonable structures, but the present stage of experimentation makes us prefer to include general matrices and linear systems in the mathematical library category.

From today's vantage point, it appears that mathematical laboratories have achieved a secure footing; that they provide a number of challenging computational-mathematical-design problems for interested students of modern computer sciences and mathematics.

Acknowledgement

The national government, through the Advanced Research Projects Agency, the National Science Foundation, the Office of Naval Research, and the Rome Air Development Center, have provided support to a number of the experimental mathematically oriented systems listed in the references. While this has been a small segment of the total support provided in computer research efforts, it has been a keystone of the efforts listed here. In several instances the Universities, themselves, and private industry have contributed facilities and hardware and have evidenced a lively amount of interest in these developments. At present, these systems have achieved a stage of utility that makes it important for such support to continue in order to bring a broad range of experimental usage through other campuses not necessarily involved in the early stages of these system's construction.

University of California, Santa Barbara
Santa Barbara, California

References

A. Experimental Mathematical Laboratories

1. The T.R.W. Canoga Park System
   Culler, G.J., and B.D. Fried: An On-Line Computing Center for Scientific Problems, Proc. 1963 Pacific Computer Conference of the IEEE, March 1963, p. 221.
2. The S.T.L. Space Park System
   Culler, G.J., and B.D. Fried: The T.R.W. Two station On-Line Scientific Computer and General Description, Computer Augmentation of Human Reasoning, Spartan Press, 1965.
3. The Columbia University System
   Klerer, M, and J. May: Two-Dimensional Programming, Conference Proceedings of the Fall Joint Computer Conference 1965, Spartan Books, pp. 63-73, 1965.
4. The Amtran System
   Albert, M.R., P.C. Clem, L.A. Flenker, J. Reinfelds, R.N. Seitz, and L.H. Wood: The Amtran Sampler System INstruction Manual, NASA Technical Memorandum, NASA TM X-53342, July 1966.
5. The University of California, Santa Barbara System
   Culler, G.J.: The User's Manual for an On-Line System, On-Line Computing, McGraw Hill, 1967.
6. The MAP System
   Kaplow, R., S. Strong J. Brachett: MAP- A System for On-Line Mathematical Analysis, MIT Publication, MAC TR 24, January 1966.
7. The Harvard System
   Ruyle, A.: The Development of Systems for On-Line Mathematics at Harvard, ACM Symposium on Interactive Systems for Experimental Applied Mathematics. August 1967.

B. User's Publications

1. Culler, G.J., B.D. Fried, R.W. Huff, and J.R. Schrieffer; Solution of the Gap Equation for a Superconductor, Phys. Rev. Letters, vol. 8, p. 399, 1962.
2. Fried, B.D., and G.J. Culler: Plasma Oscillations in an External Electric Field, Phys. Fluids, vol. 6, p. 1128. 1963.
3. Schrieffer, J.R., D.J. Scalapino, and J.W. Wilkins: Effective Tunneling Density of States in Superconductors, Phys. Rev. Letters, vol. 10, p. 336, 1963.
4. Johnson, Kenneth, and Marshall Baker: Quantum Electrodynamics, Phys. Letters, vol. 11, p. 518, 1963.
5. Cheng, Hung, and David Sharp: Formulation and Numerical Solution of Sets of Dynamical Equations for Regge Pole Parameters, Phys. Rev. Letters, vol. 132, p. 1854, 1963.
6. Field, E.C., and B.D. Fried: Solution of Kinetic Equation for an Unstable Plasma in an Electric Field, Phys. Fluids, vol. 7, p. 1937, 1964.
7. Bullock, D.L., : Exchange Ratio in CuF_2 $2H_2O$ TRW/STL Report 9891-6001-RU-000, April 1965.
8. Dixon, W.J., : ΔV to Enter Orbit About Mars, TRW IOC VM-2, April 1965.
9. DeNuzzo, J.: On-Line Solution of 2-D Trajectory Equations, TRW/STL Report 9801-6013-TU-000, May 1965.
10. Nishinago, R.G.: Preliminary Design Considerations for a Gyro-damped Gravity Gradient Satellite, TRW/STL Report 8427-6005-RU-000, May 1965.
11. Margulies, R.S.: Response of a Peak-reading Instrument to a Contaminated Signal, TRW/STL Report 9990-6963-TU-000, June 1965.
12. Pate, N.C., and S.M. Zivi: An Analysis of the Efficiency of Elliots Liquid Metal MHD Energy Conversion Cycle and IT Applicability to the Power Range of 3 to 30 KWE, TRW Systems Report 9806-602-MU-000, July 1965.

13. Fried, B.D., and L.O. Heflinger: Scaling Law for MHD Acceleration, TRW Systems Report 9801-6014-RU-000, July 1965.

14. Fried, R.D., and S.L. Ossakow: The Kinetic Equation for an Unstable Plasma in Parallel Electric and Magnetic Fields, UCLA Report R-3 Plasma Physics Group, November 1965.

15. Fried, B.D. and A.Y. Wong: Stability Limits for Longitudinal Waves in Ion Beam-Plasma Interactions, TRW Systems Report 9801-6015-RU-000, August 1965.

16. Coward, D.J.: Exact Inversion of Plasma Dispersion Relations, Phys. Fluids.

17. Collins, L: The Calculation of Unpaired Electron Density on the Nucleus of Many-electrons with a Thomas-Fermi-Dirac Potential.

18. Madon, Rabi: Superperturbation and Super-perturbation Theory Bounds on Electron Hydrogen Scattering.

THE AUGMENTED KNOWLEDGE WORKSHOP

Douglas C. Engelbart
MC DONNELL DOUGLAS CORPORATION
Information Systems Group
Cupertino, California
<AUGMENT,101931>

INTRODUCTION

The story of my involvement with on-line workstations begins in early 1951, with a vision and a life-time professional commitment. Over thirty-four years of pursuit have created a lot of personal history, and the object of this historical exercise, the workstation, occupies a unique place in it.

For me, a workstation is the portal into a person's "Augmented Knowledge Workshop" -- the place in which he finds the data and tools with which he does his knowledge work, and through which he collaborates with similarly equipped workers. And further, I consider that the large system of concepts, skills, knowledge, methods, etc. on the human side of the workstation has to be taken into account, in a balanced way, when pursuing increased human effectiveness. So, my workstation-history story embraces a rather large sphere.

The task of writing an historical piece is unfamiliar enough to cause me difficulty by itself, but the associated stirring of old records and old memories has added near overwhelming burden -- dreams, events, people, stresses, pleasures, disappointments, the firsts and the failures. Now, what from all of this -- and how to organize it -- will make an appropriate "history" paper?

I could provide a solid measure of objective reporting -- events and dates, etc. Regarding the general environment that is relevant to the workstation topic, I have been an involved observer of related computer history since 1951. I watched and experienced the supportive hardware, languages and architecture evolve, witnessed the people and efforts that brought timesharing into being, and was even more closely involved with the emergence of computer networks. Through all of this, I was wholly focused on what these things could do for people at workstations. And then there was office automation and personal computers: you don't have to be an old guy to have watched these emerge, but I'm sure they looked different to me than to most.

I could also provide lots of objective reporting about the events and dates associated with the things I have caused or had a direct hand in. There seems to be a lot there that is quite relevant to this "history of the workstation" theme. It was dusty, laborious work, this process of brainstorming for candidates, culling and ordering and trying to describe them in some reasonable sequence and context.

But what I came to realize is that there is one, clearly dominant factor that underlies essentially every cause for any uniqueness that I might list for historical record. It isn't a technology, it isn't a science, and it isn't a marketing or business model. And I am going to give it dominant coverage in this paper.

As explained below, it is what I call my "Framework." It is based upon an intuitive conviction, emplanted in my head (apparently permanently) over thirty years ago, that the gains in human knowledge-work capability which we will achieve by properly harnessing this new technology will be very large. Metaphorically, I see the augmented organization or institution of the future as changing, not as an organism merely to be a bigger and faster snail, but to achieve such new levels of sensory capability, speed, power and coordination as to become a new species -- a cat.

Based upon this conviction about huge potential gains for mankind, my Framework explains for me generally where such gains are going to come from, and provides strategic principles that can help guide a conscious pursuit of these gains.

GENESIS

I was several years out of school, possessing a B.S. in EE and two years' experience during WWII (halfway through college) as an electronic technician. I was doing odd-job electrical engineering work at Ames Research Laboratory in Mountain View, California, with the National Advisory Committee for Aeronautics (NACA, forerunner of NASA). For several months I had been devoting most of my spare time to searching for professional goals; for some reason I wanted to invest the rest of my heretofore aimless career toward making the most difference in improving the lot of the human race.

I had initially dashed off in many fanciful directions, but yet managed enough interludes of

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1986 ACM-0-89791-176-8-1/86-0073 $00.75

reasonably sober thinking to build up some useful, strategic generalizations:

> Re-treading myself professionally, to become proficient and then extraordinarily productive in some new field wasn't worth considering without a significantly attractive scenario, embedded in a reasonably structured strategic framework.
>
> The high-payoff scenarios all seemed to involve creating or joining something that, however disguised, would essentially be a crusade.
>
> Crusades have many strikes against them at the outset. E.g., they don't connect to a normal source of government or business revenue; they don't have nice organizational frameworks -- you can't go out on the streets and expect to find financial, production, or marketing vice presidents; even if you accomplished the sweeping change that was the ultimate objective, chances are that in this very complex world, the side effects might be bad enough to make you wish you hadn't; etc.

Suddenly, up through all of this delightful, youthful abstraction bobbed the following clear realization:

> The complexity of the human situation was steadily increasing. Not only that, but its rate of increase was increasing.
>
> Along with the increasing complexity had come a general increase in the urgency associated with the more critical problems.
>
> If one invented a measure for each of these -- complexity and urgency -- then for a given problem, the product of its complexity measure times its urgency measure would represent a fair measure of the difficulty mankind would find in dealing with that problem.
>
> *FLASH-1: The difficulty of mankind's problems was increasing at a greater rate than our ability to cope. (We are in trouble.)*
>
> *FLASH-2: Boosting mankind's ability to deal with complex, urgent problems would be an attractive candidate as an arena in which a young person might try to "make the most difference."*
>
> Yes, but there's that question of what does the young electrical engineer do about it? Retread for role as educator, research psychologist, legislator, ... ? Is there any handle there that an electrical engineer could ...
>
> *FLASH-3: Ahah -- graphic vision surges forth of me sitting at a large CRT console, working in ways that are rapidly evolving in front of my eyes (beginning from memories of the radar-screen consoles I used to service).*

Well, the imagery of FLASH-3 evolved within a few days to include mixed text and graphic portrayals on the CRT, and on to extensions of the symbology and methodology that we humans could employ to do our heavy thinking; and also, images of other people at consoles attached to the same computer complex, simultaneously working in a collaboration mode that would be much closer and more effective than we had ever been able to accomplish.

Within weeks I had committed my career to *"augmenting the human intellect."* In a few months, I left the NACA and enrolled as a graduate student at UC Berkeley, where Professor Paul Morton had started a computer science activity (although it would be many years before universities began calling it that), and was several years along in developing the CALDIC.

Within a few years I had to accept the fact that research on any kind of interactive computer applications just wouldn't provide me with a program acceptable to the university community for PhD and later faculty pursuit. So, I settled for something else, got my PhD and went to Stanford Research Institute (SRI) where I hoped ultimately to promote support for an augmentation program.

FRAMEWORK

That was 1957. By 1959 I was lucky enough to get a small grant from the Air Force Office of Scientific Research (AFOSR, from Harold Wooster and Rowena Swanson) which carried me for several years -- not enough for my full-time work, but by 1960 SRI began pitching in the difference.

It was a remarkably slow and sweaty work: I first tried to find close relevance within established disciplines. For a while I thought that the emergent AI field might provide me with an overlap of mutual interest. But in each case I found that the people I would talk with would immediately translate my admittedly strange (for the times) statements of purpose and possibility into their own discipline's framework -- and when re-phrased and discussed from those other perceptions, the "augmentation" pictures were remarkably pallid and limited compared to the images that were driving me.

> For example, I gave a paper in 1960 at the annual meeting of the American Documentation Institute, outlining the probable effects of future personal-support use of computers, and how this would change the role of their future systems and also provide valuable possibilities for a more effective role for the documentation and information specialists <Pub-60-SpecCons>.
>
> No response at all at the meeting; one reviewer gave a very ho-hum description as ... the discussion of a (yet another) personal retrieval system. Later, at lunch during a visit to a high-caliber research outfit, an information-retrieval researcher got very hot under the collar because I wouldn't accept his perception that all that the personal-use augmentation support I was projecting amounted to, pure and simple, was a matter of information retrieval -- and why didn't I just join their forefront problem pursuits and stop setting myself apart.

Then I discovered a great little RAND report written by Kennedy and Putt <Ref-A> which described my situation marvelously and recommended a solution. Their thesis was that when launching a project of inter- or new-discipline nature, the researcher would encounter consistent problems in approaching people in established disciplines -- they wouldn't perceive your formulations and goals as relevant, they would become disputative on the apparent basis that your positions were contrary to "accepted" knowledge or methods, etc.

The trouble, said these authors, was that each established discipline has its own "conceptual framework." The enculturation of young professionals with their discipline's framework begins in their first year of professional school. Without such a framework, tailored for the goals, values and general environment of its respective discipline, there could be no effective, collaborative work. Furthermore, if such a conceptual framework did not already exist for a new type of research, then before effective research should be attempted, an appropriate, unique framework needs to be created. They called this framework-creation process the "Search Phase".

So, I realized that I had to develop an appropriate conceptual framework for the augmentation pursuit that I was hooked on. That search phase was not only very sweaty, but very lonely. In 1962, I published an SRI Report entitled "Augmenting Human Intellect: A Conceptual Framework" <Rpt-62J>. With the considerable help of Rowena Swanson, this was condensed into a chapter of a book published in 1963 <Pub-63-Frame>.

I can appreciate that these framework documents appear to many others as unuseably general and vague, but for me the concepts, principles and strategies embodied in that framework look better and better every year.

The genesis of most of what was/is unique about the products of the augmentation work can be traced back to this framework.

PROGRAM SUPPORT

I submitted many proposals before getting support to pursue the augmentation program outlined in the framework report. Among the stream of politely phrased regrets, there was one which in contrast to today's environment can provide useful perspective on the environment of 1961.

> Four high-quality civilian experts had been enlisted by one agency as a site-visit team; brain researcher, psychologist, computer expert -- and for me it was a very enjoyable day's dialog. But the later letter from the agency informed me regretfully that [paraphrased] *"... since your interesting research would require exceptionally advanced programming support, and since your Palo Alto area is so far from the centers of computer expertise, we don't think that you could staff your project adequately ...".*

When J. C. R. Licklider came from Cambridge to take over ARPA's newly formed Information Processing Techniques Office in late 1962, I was figuratively standing at the door with the Conceptual Framework report and a proposal. There the unlucky fellow was, having advertised that "man computer symbiosis," computer time-sharing, man-computer interface etc. were the new directions -- how could he in reasonable consistency turn this down, even if it was way out there in Menlo Park?

Lick moved very swiftly; by early '63 we had a project. But whereas I had proposed using a local computer and building an interactive workstation, Lick asked us instead to connect a display to the System Development Corporation's (SDC's) AN/FSQ32 computer, on site in Santa Monica, to do our experimenting under the Q32's projected new time-sharing system. (Converting the Q32 to be a time-shared machine was SDC's IPTO project.)

Later that year, our project was modified to include an online data link from Menlo Park to Santa Monica, with a CDC 160A mini-computer at our end for a communication manager, supporting our small-display workstation. For various reasons, not uncommon in pioneering ventures, that first year was very unproductive relative to the purposes and plan of our project. Lick was willing to put some more support into the direct goal (more or less as originally proposed), but the support level he could offer wasn't enough to pay for both a small research staff and some interactive computer support.

Mind you, the CDC 160A, which was the only commercially suitable mini-computer that we knew of, even though having only 8K of 12-bit words, and running at about 6 microseconds per instruction, cost well over $100K (1963 dollars). Paper tape in and out; if the system crashed, you had to load the application program from paper tape, and the most recent dump of your working file (paper tape), before you could continue. A crude, industry-standard Flexowriter (online typewriter) could be driven; otherwise it was paper-tape in and out.

What saved my program from extinction then was arrival of an out-of-the-blue support offer from Bob Taylor who at that time was a psychologist working at NASA Headquarters (then in Washington, D.C.). I had visited him months before, leaving copies of the Framework report and our proposal, and I had been unaware that meanwhile he had been seeking funds and a contracting channel to provide some support. The combined ARPA and NASA support enabled us to equip ourselves and begin developing Version 1 of what evolved into the NLS and AUGMENT systems.

Paul Fuhrmeister, and later Eugene Gribble of NASA's Langley Research Center, had to stick out their necks as successive heads of Langley's large computational division to support the direction and supervise NASA's support for our program, which continued several years after Taylor left NASA to join ARPA's IPTO office.

Our ARPA support grew and was fostered by Lick's successors -- Ivan Sutherland, Bob Taylor and Larry

Roberts. Meanwhile, the Air Force's Rome Air Development Center, at Rome, New York, began to supply supporting funds. By 1967, it was recognized that the respective contributions from ARPA, NASA and RADC represented significant parts of a coordinated program, and the other agencies began funneling their funds through RADC, which served for many years both with monitoring and managing our contracts, and providing their own significant share of support funds. Duane Stone and John McNamara provided strong support and contract liaison from RADC.

NASA support ended by 1969, and ARPA and RADC provided significant support until 1977, although from 1974 the support became ever more for supporting applications and developments for other organizations for targets formulated by others (e.g., the National Software Works) -- and the continuing pursuit of augmentation along my strategic vector virtually stopped.

THE AUGMENTATION RESEARCH CENTER (ARC)

An historically important organizational cluster emerged at Stanford Research Institute in the sixties, peaked about 1974, and was scattered in 1977 -- with a small core carrying forth in a commercial (and then industrial) environment to the present.

It grew by ones and twos from 1963, as it collected "permanent" members from the SRI technical staff, and recruited new ones from the outside. By '69 I believe we were about 18 strong; and this grew steadily until by '76 we totalled about 45. In 1973 we made two explicit sub-groups, one headed by Dick Watson doing development of software (and some hardware), and one headed by Jim Norton handling operations and applications support.

SRI was organized by divisions, each containing a group of laboratories; the hierarchy being formed according to the associated disciplines. ARC grew to laboratory size and status, but it became something of a problem for SRI.

Other laboratories (at least in science and engineering) operated more or less as a "farmers' market," where small and changing clusters of researchers promoted and conducted research projects as a loose federation. The management structure, budgeting, accounting and financing for the Institute had evolved to support this kind of business.

But ARC was driven by a coherent, long-term pursuit. This involved the continuing evolution of an ever-larger and more sophisticated system of hardware and software. It also came to involve delivering solid support service to outside clients to provide meaningful environments for learning about the all-important co-evolution processes in human organizations (human system & tool system).

It didn't seem unreasonable to me to pursue this course: things similar and on a grander scale are common for other researchers. E.g., it is taken for granted for funding agencies to build and operate accelerators and observatories in support of research in nuclear physics and astronomy -- or to outfit ships and airplanes to support research expeditions. But, whatever my perception, there were some significant problems and stresses for which our over-all environment didn't have effective ways to cope.

In the particular dynamics involved here, there were probably seven relevant parties: me; the ARC staff; other SRI researchers; SRI management and administration; ARC's sponsors; ARC's utility-service clients; and other groups of researchers outside of SRI.

It would be an interesting historical study to try to understand the diversity of perception that must have existed among this set of players. What did the different parties perceive for the future of workstations, for the range of function and application that would come about, for the systems architectures and standards that must emerge, and for the impact on the organizations that learned how to harness these most successfully?

Even as a central party in what happened, I've not understood the dynamics. But I am pretty sure that disparities among the perceptions of all of the above parties had a major part in what to me was the "great collapse of SRI-ARC." Even if I had done everything right over the years (a laughable hypothesis), it is now fairly clear to me that it isn't the market's fault if someone fails in trying to sell it something that it isn't ready for. In other words, I can't blame those other groups. (Which of course makes for a personal problem, since during those times of black discouragement when one wants desperately to blame someone, there is only one candidate -- that guy at the head of the list).

In 1977, SRI judged it better to move our large-system development and external-service activities out from the research institute environment and into a suitable commercial environment. They advertised, entertained prospective bidders, made a selection, and negotiated a transfer of the business to TYMSHARE, Inc., of Cupertino, California. The system was renamed AUGMENT, and marketed as part of TYMSHARE's integrated Office Automation services. In 1984, McDonnell Douglas Corporation acquired Tymshare, and the small AUGMENT business is now operated as the Augmentation Systems Division of the Computer Systems Company within the MDC Information Systems Group.

A CHRONOLOGY OF EVENTS

1962

* launched a long-term R&D program with the following basic Framework Principles (see <Rpt-62J>):

  There are many inventions, skills, methods, working conventions, organizational modes, etc.

which have been integrated into the human's knowledge-work environment to "augment" his basic, inherited, biological capabilities. Let a working assemblage of all of these be called an Augmentation System; my Framework considers this whole system to be a valid object of study and improvement.

I broke the many parts of the Augmentation System into two main sub-systems: one contained all of the hardware, software and other artifacts -- the Tool System; and all the rest of it I called the Human System. Note that the Human System contains our natural languages, and the conceptualizations and formalisms of every discipline: an overwhelming network of invention. (Sometimes, in the early years, I called these the Service System and the User System).

The emergent information technologies promised such startling innovation in the Tool System that, according to my Framework, most of the Human System elements that are involved in our knowledge work are likely to be up for improvement or replacement. And the Framework further predicted very large improvements in human knowledge-work capability as a result -- after we have gone through a few generations of evolution.

The challenging strategic question for me was: how best to invest whatever resources are available in a manner that best serves the evolution toward truly high performance by our knowledge organizations?

One answer: begin consciously scouting for elements in our Human System that are candidates for being changed as a means of better harnessing the technology toward our human ends.

This led directly to such things as our structured-text, with its links and views, and also to the mouse and the keyset.

"Bootstrap" leverage was another answer. Look for the innovations that will boost not only our regular productive capability, but will add as much as possible to our capability to further improve our Augmentation System.

This kept me focused on documents -- which carry the knowledge, plans, arguments, etc. that are critical to helping us better climb the evolutionary hill. It also very directly pointed to the importance of developing improved support for collaboration among distributed workers; and from there to community support, etc.

The whole Augmented Knowledge Workshop concept emerged from the Bootstrap strategy -- working toward coherent, integrated access to an open-ended, evolving collection of resources and people. And doing this in a way that best enables evolutionary freedom of parts of the whole system without being unnecessarily anchored by the needs of other parts. From this arose such things as the Procedure Call Protocol, the application-independent User Interface System, and the Network Virtual Terminal, etc.

But read on ...

1963

* developed the chord keyset as a one-handed alternative for character input in an interactive environment. See <Rpt-65C> and <Pub-67-DispSel>.

* I participated in the ARPA-sponsored "summer study group" that kicked off Project MAC at MIT. Memories:

  Wrote a project memo about the dichotomy between "User System" and "Service System." Essentials are described in <Rpt-65C>, and the expanded dichotomy recurs as for instance in <Pub-73-AKW>, <Pub-78-IntEvolOAS>, and <Pub-82-HiPerfKW>.

  Had my turn to address the whole group (50 or so people); not much reaction, but I do remember the group hoot when I said that we'd all be seeking to boost response time for many man-computer interactions down to at least a quarter of a second before the returns began to diminish.

* tried unsuccessfully to develop a workable system using a long-distance data link from Menlo Park to Santa Monica, with the SDC Q32 as a time-sharing host, and a CDC 160A as local communication manager and display driver.

1964

* made explicit strategic decision to bypass the online typewriter and go directly to display workstations. In spite of most other online developers working with typewriters, I felt that the much higher "augmentation potential" of displays warranted early pursuit, and figured that cost would surely come down in a few years. Keep in mind that CRT workstations were very expensive then -- they weren't a consumer product by any means. The single, large-CRT workstation we attached to the CDC-3100 that year probably cost us over $100,000, and required a lot of custom work on our part. [But we have always kept an access mode open for "dumb terminals" and "glass teletypewriters," which can elicit as much service as possible within their limitations.]

* moved to stand-alone CDC 160A; paper-tape I/O to and from a Flexowriter paper-tape typewriter; first, primitive version of our online, structured-file editing. Had a simple, batch system, too: type in correction directions on a paper-tape-punching Flexowriter; feed that paper tape into the 160-A for pre-processing; results and the file to be corrected transferred via mag tape to SRI central processor (Burroughs 220, I think); processed results brought back for post-processing and printing on the 160A.

1965

* developed the mouse as part of an explicit search for optimum screen-selection techniques in association with our online-application framework. See <Pub-67-DispSel>

* moved NLS to a stand-alone CDC-3100, with online disk pack, 16K of 24-bit memory, line printer, paper-tape and punched-card I/O, custom built display. Full structured files, with in-file addressing and uniform text- and structure-manipulation commands.

* Published <Rpt-62J>, which included the following:

  Discussion of how the computer can give aid to the basic communication processes between the human and his external environment (which environment of course includes his kit of computer tools). Clear forerunner of our later User Interface System.

  Uses example of introducing a chord keyset into a user's environment to bring out specific examples of the NEED-POSSIBILITY REVERBERATION phenomena which was introduced in 1962, <Rpt-62J>, and which is a basic part of all my subsequent strategic thinking.

  Extensive discussion of USER-SYSTEM, SERVICE-SYSTEM DICHOTOMY, including the characteristics of research in each domain. (CO-EVOLUTION without yet naming it such.)

  Detailed description of the earliest version of our later FLTS (oFf-Line Text System), and what eventually became our DEX System (Deferred EXecution). Working at a Model-33 Teletype, which makes a punched-tape record of all that is typed. Escape and Command codes embedded anywhere in the text could cause the later "batch" process to correct any previous error or omission, including those in any prior commands. The Model-33 had only one case of alphabetic characters; this process enabled me to designate cases so that the later, processed printout on a two-case Flexowriter (paper-tape driven) would come out with proper alphabetic cases.

    (I made myself write most of that report this way. And my wife made me move out into the garage to do it because the Model-33 was so noisy. I remember the extra problem of typing with cold fingers.)

* Showed real time movie of NLS to ARPA IPTO Principal Investigators at our May 16-17 meeting at MIT. Simple, structure-text manipulations, with very fast concurrent control (mouse and keyset), and very fast computer response (stand-alone CDC-3100). Illustrated the basic difference in perspective between our approach and the prevailing concepts of "time-shared computer support." At that evening's cocktail hour, Bob Taylor told me, *"The trouble with you, Doug, is that you don't think big enough."*

Well, after he dragged out of me a description of what I'd really like, he encouraged me to formulate a proposal for it -- a multi-user laboratory, based on a time-shared machine, to get on with real bootstrapping.

1966

* all source-code development, maintenance and documentation now fully moved into the NLS environment (providing natural and powerful support for structured programming).

* began planning for move to time shared environment;

1967

* at Spring meeting of ARPA Principal Investigators, in Ann Arbor, during session kicking off ARPA's newly revealed Network development project, I volunteered to develop and operate a Network Information Center (NIC) -- as a special, prototype support service for the network-connected community of ARPA-IPTO researchers.

* developed our first explicitly separated command processor (I used the term "control" then, instead of "command"), supported with our "Control Meta-Language," a separate command-description, compileable language. Cf. Branch 4c and Fig. 9 of <Pub-68-ResCen>.

* developed our MOL 940, a Machine Oriented Language and compiler for the SDS 940.

* equipped a meeting room with specially-built tables, video hookup, and computer controls for real-time support of meetings with our CDC-3100 version of NLS. (Photo and description in <Rpt-68D> and in <Ref-B>.)

  Over twenty people could participate, each had full view of a nearby video screen; each could pick up a mouse to control a specially shaped "participant's cursor" so all could see what he was pointing at; one master control station provided discussion-recorder control for accessing, flexible viewing, and modification of project documents, agendas, formulation of measures to act upon, etc.

  We used this support system on 12-13 October, 1967, for a progress-review meeting with our research sponsors: Bob Taylor and Barry Wessler from ARPA, Fred Dion and Dean Bergstrom from RADC, and Gene Gribble from NASA.

1968

* NLS by now had a full set of basic features that have since characterized it and AUGMENT (its commercial successor). E.g., full-screen, integrated outline and text processing; in-file addressing; in-file and cross-file text or structure manipulation by address; basic repertoire of view-control commands; content filtering; generalized, computer-executable citation links; verb-noun, consistent command syntax with optional use of ultra-fast,

concurrent control using the mouse and chord keyset;

Included also was a calculator package, integrated into NLS: mouse-selecting operands; totaling columns; inserting accumulator contents at selected locations or replacing selected numbers in a file; executing user macros with pauses and prompts for users to select file variables or provided typed in values.

* Put together our home-designed, custom-built display system to run with the SDS-940.

Two custom-built, random-deflection display generators were each time-shared to drive six, small, 5" high-precision CRTs. In front of each of these CRTs was mounted a high-quality, video camera so as to scan the CRT face. These twelve video lines were brought out to our work area, where each work station had a high-quality video monitor for its display.

This gave us four sizes of alphanumeric characters, and accelerators vector-graphic figures. The display generators were connected on a Direct Memory Access bus so that switching from one stored view to another occurred essentially in a thirtieth of a second.

* came online with an improved NLS on a time-shared, SDS 940; large swapping drum; special, home-made display system operating from direct-memory access, providing integrated text and graphics, and delivering video to up to twelve workstations out in our laboratory.

* Made a public debut at the Fall Joint Computer Conference in San Francisco, December. For this event, we added another layer of new technology on top of NLS, a system that was already very complex for its day. It is worth an extra bit of description here.

Bill English and I wrote a paper for this conference describing ARC's objectives, physical laboratory, and the current features of NLS. In the Spring, when the Program Committee was considering candidate papers and organizing its sessions, I also proposed that they let us have a full hour-and-a-half session to put on a video-projected, real-time presentation. After considerable deliberation, and no less than two site visits to our lab at SRI, they consented.

It was a considerable gamble, possibly an outright misuse of research funding. I have no illusions that it could possibly have been pulled off without Bill English's genius for getting things to work. Our new display system provided us with twelve video cameras; we left about half of them working as display generators, and used the others to provide video views of people, borrowing tripods and drafting all kinds of people as camera operators and prompters.

We leased two video links to send images from SRI to the Conference Center in San Francisco -- a direct distance of about 30 miles. It required temporarily mounting four pairs of dishes -- two atop our SRI building, two atop the Conference hall, and four on a truck parked on top of a relay mountain. We procurred some video-lab equipment: frame splitters, switches, faders, and mixers. We made special electronics to get our mouse and other terminal signals from the podium to the 940 at SRI.

It required a special video projector, whose rental included a specialist from New York to set it up and operate it. He proved invaluable in making other things work that day, too. Two cameras were mounted on the stage where I sat at the special work station (which the Herman Miller Company had made for us, and donated).

I was on-stage as anchor man during the continuous, 90-minute presentation, and Bill sat in the canvas-enclosed, raised booth at the back of the auditorium, directing the participants according the the script that I had prepared. People in our laboratory had key roles, and Bill coordinated us all via a voice intercom, while he also did the switching and mixing and frame splitting to put together the projected images.

During that 90 minutes, we used the projected display images (composite text & graphics) both to present agendas and descriptive portrayals, and also to demonstrate what NLS could do and how we applied it to our planning, documenting, source-code development, business management, and document retrieval.

The audio and video of the entire presentation was captured on film (no portable video recorders in those days). We had ten prints made, and circulated free loaners to people for years. (This film has recently been converted to VHS video cassette form to facilitate viewing by other people.)

1969

* began design of windowing capability for NLS.

* developed concept of a user "reaching through" his personal workplace (i.e., his familiar online working files and application programs) to access less basic, specialized data and application processes (and other people); i.e., the "reach through" should provide access to these, translated by the integrated support system, so as to appear as coherent parts of his familiar, personal workplace.

* specified our first mail and "Journal" system as part of an explicit pursuit of a "Dialog Support System," planning for it to be part of our ARPANET-NIC service.

* Developed document-outputting capability processing our composite, text-graphic document files to drive a service-bureau, CRT-based, full-page, Stromberg-Carlson photo printer to produce documentation with graphics and text mixed on the same pages.

* Became the second host on the ARPANET with our SDS 940. (UCLA was first, UCSB next, then the University of Utah, then ....)

1970

* (See also <Rpt-70D> and <Rpt-72F>.)

  Detailed use of NLS for internal management processes of ARC: Cost records, working forecasts, purchase Renaissance, etc.

  In 1970 we began using the ARPANET to facilitate our re-programming of NLS for the forthcoming PDP-10 TENEX. The University of Utah had a TENEX on the network, and we used NLS on the 940 to write our new PDP-10 code; using our Tree-Meta compiler, we developed a cross-compiler for our 940 that produced PDP-10 relocatable binary code. We would ship that over the net for loading and debugging on Utah's TENEX.

  When the two computers and the intervening network link were all working properly (lots of flat tires in the early days of automobiles), our programmers would do all of this back and forth transitioning "through" the same workstation.

  I think that it was not only a record-making way of working, but the NLS transport task was accomplished in remarkably short time (we attributed part of the efficiency to the network, and part to the use of NLS).

  In late 1970 we brought NLS up on the PDP-10 TENEX with improved and new features (including multiple windows).

  The transfer process, and a detailed description of the design changes and new features for NLS are described in <Rpt-71F>.

* began using our Mail/Journal system within our group. Integrated into NLS, this assumed that a mail item was a document -- so any part of all of an NLS document could be sent. Provided for permanent record in explicitly retrievable form (our Journal). As an electronic-mail system, this was quite advanced. It had a Directory service (our Ident System) to provide mail-relevant information about registered users; mail distribution was addressed by people's Idents, with no need to know or specify which host they used. Fields were provided for superceding other items, and for attaching keywords. An online index was provided for stored items. See descriptions in <Rpt-71F> and in <Pub-75-TelConf>.

1972

* began developing our first, integrated Help system.

* formulated the "AKW Architecture" -- implemented in stages, described in <Pub-82-HiPerfKW>.

* implemented the "shared-screen," televiewing mode of online collaboration between two or more NLS users. See <Pub-68-ResCen>, <Pub-75-TelConf> and <Pub-78-IntEvolOAS>.

1973

* brought up a Table Subsystem in NLS.

* designed our first, totally modular User Interface System, as later described in <Pub-82-HiPerfKW>. Got it running on a PDP-11 that talked to our TENEX through the network, via our Procedure Call Protocol.

* Developed our Line Processor, as described by Don Andrews in <Pub-67-DispSel>. It incorporated Intel's first microprocessor (the 4004) in a special box which was inserted in the communication line between a dumb display terminal and a modem. This made use of our Virtual Terminal Protocols, and managed a multi-window, two-dimensional screen using off-the-shelf, "dumb" display terminals. Our mouse and keyset input devices were plugged into the line processor, which appropriately translated their actions to control cursor position and special communications to the host. A printer port on the line-processor provided local printout service; a special communication protocol allowed the host to send printer packets mixed in with display-support packets.

* Finalized specification for our Network Virtual Terminal, something which has become a key part of our architecture. The objective on the one hand was to free the application programmers from worrying about the special features of different workstations, and on the other hand, to enable more flexible evolution by users of workstations they may adopt to fit particular needs. As part of this, there was a terminal-independent Display Manipulation Protocol for communication from application program to terminal, and an application independent Input Protocol for communicating from terminal to application program.

* generalized the file structure of our document files to provide for generalized property structures associated with each addressable object; intended to accommodate composite integration of such as graphics, digitized speech, scan-coded images, or any other arbitrary data form.

1974

* gave up our high-performance, local display system for the line-processor supported, remote display system -- to make ourselves live with the same remote services as our NIC clients and Utility customers. [On principle, we gave up our integrated, direct-view graphics and the fast response of our direct-memory-access, local display generator.]

* opened our "Workshop Utility Service" -- delivering NLS service over the ARPANET to DoD customers as pilot applications of office

information service; had gone out on bid for commercial time-sharing services, selected Tymshare Inc. of Cupertino, Ca.; their host, named Office-1 provided the computer service; we fielded special trainers and application development staffs and cultivated special customer representatives into a spirited community.

1975

* implemented our new, integrated graphics system, which could support remote display and manipulation of illustrative graphics on a Tektronix 4014 storage-tube display plugged into the line-processor's printer port. The graphic data were embedded within the NLS structured-text files; an illustration produced as a plotter-driver file by any other graphics system could be picked up and attached to a specified location in an NLS document, and be subsequently viewed and modified. Also, our Output Processor could direct that a properly scaled image of a each such illustration be located appropriately within a multi-font page layout.

CONCLUSIONS

I hadn't realized just how much history there is back there in my past. I wish that I could have done a more uniform job of digging and organizing for this paper. Oddly enough, a major problem is that I am quite distracted by the needs and opportunities for working on tomorrow's history -- there is still such a large "unfinished agenda" stemming from my compulsive commitment to the Augmentation Framework.

Re-living the past during this history exercise has left me with a feeling of having re-visited the Dark Ages. Paper tape, expensive technology, and years of frustrations with flaky, first-of-a-kind systems -- thank God that we have now emerged into the Rennaisance.

But another side of my experience has been with what people perceive about the possibilities stemming from these new technologies. And I am not so sure that we have moved all that far out of the Dark Ages in that regard.

I still don't see clear perceptions about what we humans can gain in new capabilities, or about how this may come about. There are constant, echoing statements about how fast and smart the computers are going to be, but not about how the enhanced computer capabilities will be harnessed into the daily thinking and working life of our creative knowledge workers.

I guess what I am hoping to see is the emergence of professional societies concerned with something like what I call a whole Augmentation System. And I'd like there to be special conference sessions and research efforts focused on pursuit of very high performance for teams of people equipped with integrated new technologies AND with new concepts, methods, roles, skills, etc.

And I'd like to hear people begin talking about the new OPPORTUNITIES to change their ways and skills so that they can more EFFECTIVELY HARNESS the new tools, in pursuit of their own INCREASED EFFECTIVENESS. I very much look forward to harnessing smarter and smarter computer processes within my working domain, but I would like to hear less about making computers very smart so that (apparently) the user won't have to learn new things.

PARTIAL CAST OF CHARACTERS

We recently had a reunion of all the people that had worked with NLS/AUGMENT over the years. We found about 150 names when we dug back in records and people's memories. Many of those people gave spirited support and since have gone on to work at other kinds of jobs. And many have stayed in a similar job area and have continued to contribute elsewhere -- publishing, developing, entrepreneuring and consulting.

Here is an alphabetic listing of some of these latter, stemming from the more historical era:

Don Andrews
Jim Bair
Roger Bates
Bob Belleville
Dave Casseres
Jan Cornish
Tom Davis
Chuck Dornbush
Bill Duvall
Joe Ehardt
Bill English
Jake Feinler
Larry Garlick
Martin Hardy
Ken Harrenstein
Dave Hopper
Charles Irby
Susan Kahn
Kirk Kelley
Harvey Lehtman
Rob Lieberman
Dave Maynard
John Melvin
Dean Meyer
Elizabeth Michael
Ray Panko
Bruce Parsely
Bill Paxton
Jeff Peters
Andy Poggio
Jon Postel
Raphi Rom
Jeff Rulifson
Glenn Sherwood
Dave Smith
Jacques Vallee
Dirk Van Nouhuys
Ken Victor
Smokey Wallace
Dick Watson
Anne Weinberg (Duvall)
Jim White

REFERENCES

NOTE: Arrangements are being made to collect all of the records and films from the SRI-ARC era and store them in the History-of-Science archives at the Stanford library. This will include all of the Journal items generated by ARC staff, which is a very unique collection of detailed history beginning in 1970. These archived records will be available to qualified researchers; access will be controlled by the library archivist.

RPT: Some Relevant SRI-ARC Reports

<Rpt-62J>: "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Stanford Research Institute, on Contract AF 49(638)-1024, October 1962, 134 pp, Engelbart, D. C.

<Rpt-65C>: "Augmenting Human Intellect: Experiments, Concepts, and Possibilities," D. C. Engelbart; Summary Report, Stanford Research Institute, under Contract AF 49(638)-1024 for Directorate of Informtion Sciences, Air Force Office of Scientific Research, March 1965. SRI Project 3578; 65 p. <XDOC, 9691,>

<Rpt-65G>: "Computer-Aided Display Control," W. K. English, D. C. Engelbart, Bonnie Huddart, SRI Final Report to NASA Langley Research Center under Contract NAS 1-3988, July 1965, 104 p. <XDOC, 9692,>

<Rpt-68D>: "Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," D. C. Engelbart, W.K. English, J.F. Rulifson, SRI Final Report for AF Rome AIr Development Center under Contract AF 30(602)-4103, April 1968, nnn p., <XDOC, 9697,>

<Rpt-70D>: "Computer-Augmented Management-System Research and Development of Augmentation Facility," D. C. Engelbart, Final Report, RADC, <AUGMENT, 5139,>

<Rpt-71F>: "Network Information Center and Computer Augmented Team Interaction," Technical Report RADC-TR-71-175, by D. C. Engelbart as principal investigator for SRI Contract F30602-70-C-0219 under ARPA Order No. 967. June 1971, 99p. <AUGMENT, 8277,>

<Rpt-72F>: "Online Team Environment: Network Information Center and Computer Augmented Team Interaction," D. C. Engelbart, Final Report RADC-TR-72-232 to Air Force Rome Air Development Center for SRI work under ARPA Order No. 967., June 8, 1972, 264 p. <AUGMENT, 13041,>

PUB: Open-Literature, "Augmentation" Publications

<Pub-60-SpecCons>: "Special Considerations of the Individual as a User, Generator, and Retriever of Information," American Documentation, 12, No. 2, pp. 121-125, April 1961. [Paper presented at the Annual Meeting of the American Documentation Institute, Berkeley, Californaia, October 23-27, 1960.], Engelbart, D. C.

<Pub-63-Frame>: "A Conceptual Framework for the Augmentation of Man's Intellect," in Vistas in Information Handling, Howerton and Weeks (Editors), Spartan Books, Washington, D. C., 1963, pp. 1-29. Engelbart, D. C.

<Pub-67-DispSel>: "Display-Selection Techniques for Text Manipulation," IEEE Transactions on Human Factors in Electronics, Vol. HFE-8, No. 1, March 1967, Pp. 5-15. English, William K., Douglas C. Engelbart and Melvyn L. Berman.

<Pub-68-ResCen>: "A Research Center for Augmenting Human Intellect," AFIPS Conference Proceedings, Vol. 33, Fall Joint Computer Conference, San Francisco, December 1968, pp. 395-410 FJCC 1968. <AUGMENT,3954,>, Engelbart, D. C., W. K. English

<Pub-70-IntImpl>: "Intellectual Implications of Multi-Access Computer Networks," Proceedings of the Interdisciplinary Conference on Multi-Access Computer Networks, Austin, Texas, April 1970 <AUGMENT,5255,>, Engelbart, D. C.

<Pub-72-CommServ>: "Coordinated Information Services for a Discipline- or Mission-Oriented Community," Proceedings of the Second Annual Computer Communications Conference in San Jose, California, Jan 24, 1972 -- also, Published in "Computer Communication Networks," R. L. Grimsdale and F.F. Kuo editors, Noordhoff, Leyden, 1975. <AUGMENT,12445,> Engelbart, D. C.,

<Pub-73-AKW>: "The Augmented Knowledge Workshop," AFIPS Conference Proceedings, Volume 42, pp. 9-21, National Computer Conference, June 4-8, 1973. Engelbart, D. C., R. W. Watson and J. C. Norton. <AUGMENT,14724,>

<Pub-73-KWTermDes>: "Design Considerations for Knowledge Workshop Terminals," AFIPS Conference Proceedings, Volume 42, pp. 221-227 , National Computer Conference, June 4-8, 1973. Engelbart, D. C., <AUGMENT,14851,>

<Pub-74-LineP>: "Line Processor -- A device for amplification of display terminal capabilities for text manipulation," AFIPS Conference Proceedings, pp. 257-265 , National Computer Conference, 1974. <AUGMENT,20184,> Andrews, Donald I.,

<Pub-75-TelConf>: "NLS Teleconferencing Features: The Journal, and Shared-Screen Telephoning," Engelbart, D. C., CompCon75, Sept. 9-11 '75, Digest of papers, pp.173-176. [IEEE Catalog No. 75CH0988-6C] <AUGMENT,33076,>

<Pub-76-HLFrame>: "A High-Level Framework for Network-Based Resource Sharing," James E. White, J. E., AFIPS Conference Proceedings, June 1976, NCC, Volume 45, pp. 561-570. <AUGMENT,34263,>

<Pub-76-UIDes>: "User Interface Design Issues for a Large Interactive System," Richard W. Watson, AFIPS Conference Proceedings, Volume 45, AFIPS Press, 1976, Montvale, NJ, pp. 357-364. <AUGMENT,27171,>

<Pub-77-DAD>: "The Design and Implementation of DAD, A Multiprocess, Multimachine, Multilanguage Interactive Debugger," Victor, K. E., Proceedings of the Tenth Hawaii International Conference on System Sciences, University of Hawaii, 1977, pp. 196-199. <AUGMENT,29079,>

<Pub-77-SEngrEnv>: "A Software Engineering Environment," Kenneth E. Victor, Proceedings of AIAA/NASA/IEEE/ACM Computers In Aerospace Conference, Los Angeles, CA, October 31-November 2, 1977, pp. 399-403. <AUGMENT,29292,>

<Pub-78-IntEvolOAS>: "Toward Integrated, Evolutionary Office Automation Systems," Proceedings of the Joint Engineering Management Conference, Denver, Colo., 16-18 Oct 78, pp 63-68. <AUGMENT,71279,>, Engelbart, D. C.

<Pub-80-EvolOrg>: "Evolving the Organization of the Future: A Point of View," A paper published in "Emerging Office Systems," Proceedings of the Stanford International Symposium on Office Automation, March 23-25, 1980, edited by Robert Landau, James Bair, Jeannie Siegman, Ablex Publications Corporation, Norwood, NJ. <AUGMENT,80360,>, Engelbart, D. C.

<Pub-82-HiPerfKW>: "Toward High-Performance Knowledge Workers," Douglas C. Engelbart, OAC'82 Digest, Proceedings of the AFIPS Office Automation Conference, San Francisco, April 5-7, 1982, pp. 279-290. <AUGMENT,81010,>

<Pub-84-AuthSupp>: "Authorship Provisions in AUGMENT," Douglas C.Engelbart, COMPCON '84 Digest, Proceedings of the 1984, COMPCON Conference, San Francisco, Ca., February 27 - March 1, Pp. 465-472. <OAD,2250,>

<Pub-84-CollabSupp>: "Collaboration Support Provisions in AUGMENT," Douglas C. Engelbart, OAC '84 Digest,Proceedings of the 1984, AFIPS Office Automation Conference, Los Angeles, Ca., February 20-22, Pp. 51-58. <OAD,2221,>

REF: Other-Author Publications

<Ref-A>: "Administration of Research in a Research Corporation," J. L. Kennedy and G. H. Putt; RAND Corporation Report P-847 (20 April 1956)

<Ref-B>: "The Computer as a Communication Device," from J.C.R. Licklider and Robert W. Taylor, with Evan Herbert ed., Science & Technology, April 1968, pp. 21-31.

<Ref-C>: "As We May Think," Vannevar Bush; The Atlantic Monthly (July 1945)

<Ref-D>: "Man-Computer Symbiosis," J. C. R. Licklider; IRE Transactions on Human Factors in Electronics (March 1960).

<Ref-E>: "On-Line Man-Computer Communication," J. C. R. Licklider and W. E. Clark; Proceedings Spring Joint Computer Conference, Vol. 21, pp. 113-128 (National Press, Palo Alto, California, May 1962).

<Ref-F>: "Solution on Non-Linear Integral Equations Using On-Line Computer Control," G. J. Culler and R. W. Huff; paper prepared for Spring Joint Computer Conference, San Francisco, May 1962. [Assumedly published in Proceedings Spring Joint Computer Conference, Vol. 21, (National Press, Palo Alto, California, May 1962).]

<Ref-G>: "Augmenting Your Intellect" -- Editorial interview with Douglas C. Engelbart, in Research/Development (Magazine), August 1968, pp. 22-27.

<Ref-H>: "Toward the Decentralized Intellectual Research/Development (Magazine), Article by Nilo Lindgren, staff writer, in Innovation, No. 24, September 1971. (XDOC, 10430,)

<Ref-I>: "The Origin, Anatomy, and Varieties of Mus Computeralis," (It's the Year of the Mouse!), by Tom Foth, Softalk, Vol. 2, April 1984, Pp.88-96.

<Ref-J>: "Of mice and Men," Steven Levy, Popular Computing magazine, May 1984, pp. 70, 75-78.

<Ref-K>: "The Loneliness of a Long-Distance Thinker," Chapter Nine, pp. 174-204, in book "Tools for Thought: The People and Ideas behind the Next Computer Revolution," by Howard Rheingold, Simon & Schuster, New York, 1985, 335 pp.



The Dynabook - Past, Present, and Future

Alan Kay

Apple Computer, Inc.

Paper not received in time for inclusion in the Conference Proceedings.
A full paper will be distributed at the Conference.

Personal Distributed Computing: The Alto and Ethernet Hardware

Charles P. Thacker
Systems Research Center
Digital Equipment Corp.
130 Lytton Avenue
Palo Alto, CA 94301

Abstract

Between 1972 and 1980, the first distributed personal computing system was built at the Xerox Palo Alto Research Center. The system was composed of a number of Alto workstations connected by an Ethernet local network. It also included servers that provided centralized facilities. This paper describes the development of the hardware that was the basis of the system.

Introduction

In the last few years, a new type of computing environment has become available. These *distributed personal computing* systems represent another step in the process, started by timesharing, of bringing computing power closer to the user. Although many variations are possible, these systems share a number of characteristics:

- They are based on *workstations*—personal machines that are sufficiently powerful to satisfy essentially all the computational needs of a single user. The workstations include high resolution displays, and provide a highly interactive user interface.
- The workstations are interconnected by *local networks* that provide high bandwidth communication throughout a limited area, typically a single building.
- In addition to the workstations, the network contains *servers*—nodes that provide capabilities that need to be shared, either for economic or logical reasons.

Timesharing systems grew primarily from the vision of man-computer symbiosis presented by J.C.R. Licklider in a landmark 1960 paper [29]. Efforts to realize the possibilities presented in this paper occupied the creative talents of many computer science researchers through the sixties and beyond. Distributed personal computing systems build on this view of the way computers and people interact by providing a level of responsiveness that timesharing systems cannot achieve.

The first distributed personal computing system was built at the Xerox Palo Alto Research Center over a period spanning 1972 to 1980. The workstation used in this system was the Alto [36]; the network was Ethernet. This paper describes the hardware that was the foundation of this system. A companion paper by Butler Lampson [24] describes the software that was built on the hardware base described here.

This paper contains seven sections: Section 1 describes the environment in which the work was done. Section 2 traces some of the underlying ideas. Section 3 describes the early implementation period, and section 4 discusses the servers that provide printing and file storage in the system. In section 5, the reengineering effort that made the Alto into a successful internal product is described. Section 6 briefly discusses some of the Alto's successors, and section 7 contains concluding remarks.

1 The Environment

The Xerox Palo Alto Research Center (PARC) was established in 1970, primarily through the efforts of Jacob Goldman, director of corporate research. It was composed of three laboratories: the Computer

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0087 $00.75

Science Laboratory, the System Sciences Laboratory, and the General Science Laboratory. To direct the new center, Goldman recruited George Pake, a physicist who was at that time provost of Washington University, St. Louis.

To establish the Computer Science Lab, Pake engaged Robert Taylor, who had directed the Information Processing Techniques Office of ARPA during the late sixties. Taylor had worked with and funded many of the leading computer science research groups during this period. As a result, he was in a unique position to attract a staff of the highest quality.

During the first year of CSL, Taylor built a group of approximately fifteen researchers, drawn from MIT, the University of Utah, and CMU. Several members of CSL came from Berkeley Computer Corporation, a start-up company composed primarily of individuals from the University of California at Berkeley, who had built one of the first timesharing systems [25]. From Bolt, Beranek and Newman, Taylor later brought Jerome Elkind, who was the manager of CSL from 1971 until 1976. Also during this period, Alan Kay, who was to provide much of the vision on which the Alto was based, was recruited into the Systems Sciences Laboratory by Taylor. Kay established the Learning Research Group (LRG), and defined its goal: To produce a programming system in which "...simple things would be simple, and complex things would be possible [20]."

The research environment built by Taylor was one of the main reasons for the success of CSL and its projects over the next decade. Unlike other PARC laboratories, CSL was not organized into permanent groups. Instead, researchers were encouraged to move between projects as their talents and the needs of the projects dictated. This flat structure and the mobility it made possible encouraged members of the lab to become familiar with all activities. Additionally, it provided a continuous form of peer review. Projects which were exciting and challenging obtained something much more important than financial or administrative support; they received help and participation from other CSL researchers. As a result, quality work flourished, less interesting work tended to wither.

During 1971 and early 1972, most of the effort in CSL was spent in building a set of hardware and software facilities to support the future work of the laboratory. The Maxc timesharing system [13] was built and the Tenex [2] operating system was acquired and modified for it. Projects in graphics, computer networking, and language design were started.

The main research theme of CSL—office information systems—was also developed during this period. Most of the research done in CSL and SSL during the next five years was organized around this theme, which reflected the desire of Xerox to expand its traditional copier business to include most of the functions performed in offices. Eventually, this theme was broadened to include what is now known as distributed personal computing, but initially our ambitions were lower.

A strategy for carrying out work in experimental computer science was also adopted at this time. It was based on the idea that demonstrations of "toy" systems are insufficient to determine the worth of a system design. Instead, it is necessary to build *real* systems, and to use them in daily work to assess the validity of the underlying ideas, and to understand the consequences of those ideas. When the designers and implementors are themselves the users, as was the case at PARC, and when the system is of general utility, such as an electronic mail system or a text editor, there is a powerful bootstrapping effect. This view of systems research is quite different from that found in most academic environments, since it requires larger groups working over a longer period than a university can usually support.

It was clear that the ability to provide systems with high levels of functionality would be limited by software considerations far more than by the capabilities of the underlying hardware. While it would be necessary for us to build hardware, since the needed capabilities were not commercially available, the characteristics of the devices would be determined primarily by the needs of the software systems for which they were intended. This view is commonplace today, as hardware performance has increased and its cost has declined, while the cost of delivering large, reliable software systems has continued to increase. It was much more radical in 1971.

Although hardware development was an integral part of the work carried out in CSL, it represented a small fraction of the overall activity of the laboratory. At no time did the number of people engaged primarily in hardware work exceed five—roughly ten percent of the total professional staff. This core group was very effectively augmented during large projects (e.g., the development of the Dorado [27] during 1977-80) by laboratory members with computer science, rather than electrical engineering backgrounds. However, most things were done by a relatively small group. For this reason, it was necessary to be selective in our choice of projects. Simplicity and utility were the most important criteria. Highly complex designs would have been beyond our capabilities, and the construction of systems without a wide user community would not have justified the expenditure of scarce design and implementation talent.

2 Sources of Ideas

By late 1972, most of the laboratory facilities were in place in CSL, and the researchers who had produced them began planning longer-term projects. It was at this time that the main ideas underlying the Alto were discussed and refined into an actual design. Although a number of people in CSL and SSL contributed to the specification of the new system, Butler Lampson, Alan Kay, and Bob Taylor were the individuals who were primarily responsible for shaping the design. To the extent that CSL had project managers, I filled that function. My task was to convert the vision of Lampson, Kay, and Taylor into working hardware.

Taylor had originally proposed that the primary computing facility in CSL should be an interconnected collection of small display-based machines, rather than a central timesharing system. He thought that a sufficient amount had been learned about the design of interactive systems, and that hardware costs were low enough that it would be feasible to produce such a system in modest quantities. In 1970, this idea contained a number of technical difficulties. Lampson and I argued that a stable set of computing facilities as well as experience in producing hardware in the new laboratory would be required before embarking on such an ambitious project. This conservative view prevailed, but by 1972, sufficient progress had been made in a number of areas, particularly semiconductors, that the difficulties did not seem as overwhelming.

In his 1969 thesis, Alan Kay had described a small computer system, the "reactive engine" [21], that shared many of the characteristics of the new machine. Like Taylor, Kay wanted a system that would provide a complete work environment for its user, including text and graphics manipulation, computing, and communications capability. By 1972, this vision of computing had acquired a name—Dynabook—and for a while, the Alto was called the "interim Dynabook" by its developers. Kay's vision was that the ultimate Dynabook would be portable, so that it could provide all the functions provided by books, paper, pencil, and terminals. Although the Alto never achieved this part of his vision, it served for a number of years as the hardware environment for the Smalltalk system [16], in which a number of text and graphics, music, and simulation applications were built.

Lampson's view of the capabilities of the new system and its uses was perhaps the most explicit. In a 1972 guest editorial [23] in Software–Practice and Experience, he had predicted that within five years, it would be possible to build a system "... comparable to a 360/65 in computing power for a manufacturing cost of perhaps $500." In the same article, he predicted significant advances in programming technology, and foresaw the some of the effects that would follow from these developments:

> As a result, millions of people will write nontrivial programs, and hundreds of thousands will try to sell them... Almost everyone who uses a pencil will use a computer, and although most people will not do any serious programming, almost everyone will be a potential customer for serious programs of some kind.

Although it was clear that we could not achieve these goals in 1972, there was a clear consensus that the new system should have characteristics that were prototypical of this vision. Cost was not a primary consideration in the design, but it could not be outrageous, since the system had to be producible in modest quantities to justify the development of software for it.

By late 1972, the principal features of the new machine had been defined. The major departure from past systems was the machine's display (see Figure 1). To emulate as many of the characteristics of paper as possible, we chose to provide a full bitmap, in which each screen pixel was represented by a bit of main storage, and to use raster scanning rather than the lower-cost calligraphic techniques popular at the time. We were encouraged by the earlier experiences of a group in SSL, which had developed a character generator for a similar, but higher resolution display. The display resolution was 606 pixels horizontally by 808 pixels vertically, which allowed display of a full page of text. The display image was refreshed directly from main memory, so arbitrary graphics could also be produced using the machine's load and store instructions. Initially, a specialized instruction was provided to paint characters from a font in memory into the bitmap; this facility was later superseded by the more general BitBlt primitive invented by Dan Ingalls [17]. The Alto contained no support for other graphic primitives, since we were primarily interested in text and engineering drawing applications that could be done with specialized character sets and straight lines.

The decision to provide a high-performance display came directly from our view that the most important purpose of the machine and the software that would be written for it was to provide a high bandwidth interface to the human user. Timesharing systems had made computing more accessible and decreased its

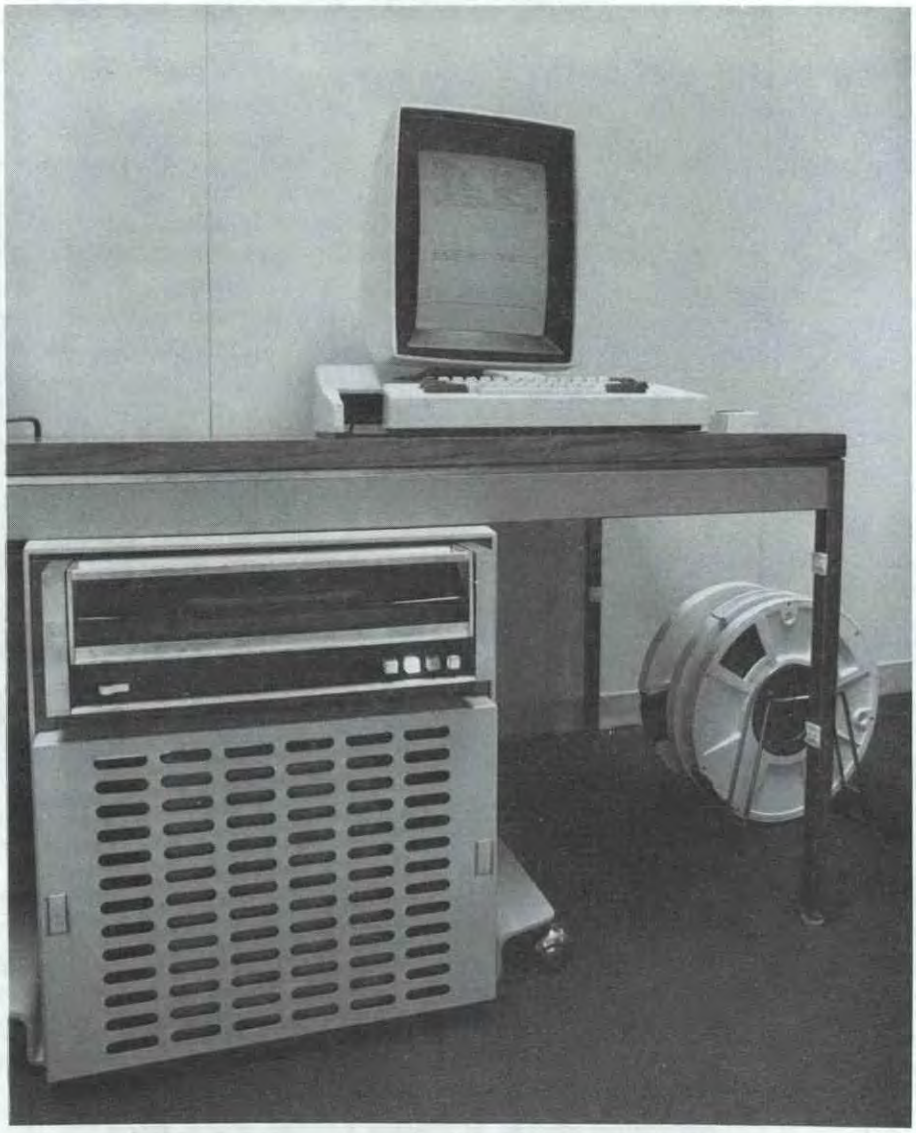


Figure 1: The Alto II Workstation. The Alto I had an identical display, keyboard, and mouse, but a slightly different cabinet.

cost, but they had done little to increase the *quality* of man-machine interaction. We viewed improvement of the user interface as extremely important, and were willing to expend a considerable fraction of the machine's resources in providing it.

Another important feature of the user interface provided by the Alto was the use of a mouse as a pointing device. This was not a PARC innovation; it had been used with considerable success in the pioneering NLS system of Englebart done at SRI in the late sixties [12,11,10]. When rolled over a work surface adjacent to the keyboard, the mouse provides relative positioning information, usually used by software to control the position of a cursor on the display screen. It also provides additional input through buttons on its top surface. Subsequent research [5] has shown that the mouse is a Fitts' Law device, in that it is as efficient for target selection as manual pointing. The practical impact of this is that in the domain for which it was intended, the mouse, iike the compact disk in the audio domain, does as well as the limits of the human user allow. This is often overlooked by those attempting to provide better pointing devices. The mouse is not as effective as a pencil or a graphics tablet for freehand drawing, but very few graphic applications made use of a tablet, although an interface for one was provided.

The display's cursor was a small image, sixteen pix-

els square, whose contents and position could be controlled by software. Many programs made considerable use of the programmability of the cursor, using its contents to convey information about the item to which the user was pointing.

The Alto keyboard was similar to that of a typewriter; it was not accidental that it lacked the cursor positioning keys and numeric keypad found on most personal computers today. In addition to the normal typing keys, it provided eight uncommitted keys that could be used by software as option or function keys. A five-finger keyset, which had been used successfully in Englebart's NLS, was provided as an enhancement to the keyboard, but it required a trained operator for use, and never became popular as an input device.

The original Alto contained 128 thousand bytes of main storage, and a 2.5 million byte cartridge disk. This was similar to contemporary minicomputers, and constituted a fairly serious error. If we had understood how rapidly semiconductor technology would advance, and how long the Alto would live, we would have included more convenient means for accessing a larger memory. We failed to do this, and although the main memory was subsequently expanded to 512 thousand bytes, it was difficult for programs to make use of the additional memory. This hampered software development quite a bit in later years.

The processor of the Alto was specified with flexibility and expansion in mind. It was microcoded, which allowed us to experiment with new instruction sets and with with new input-output devices. The principal characteristic that served to differentiate it from the minicomputers of its time was that the microprogrammed processor was shared between the emulation of a target instruction set and the servicing of up to fifteen additional fixed-priority *tasks*, most of which were associated with the machine's input-output devices. Task switching occurred rapidly, typically every few microseconds. This mechanism allowed the input-output controllers to be very simple, since they could make use of the processor for much of their work. Since access to the single-ported memory is the bottleneck in a small system, multiplexing the processor in this way did not degrade system performance. This technique had been used before on the Lincoln Laboratory TX-2 [14]; it was very successful in the Alto, and has been used in several of the Alto's successors.

To be an effective replacement for centralized computing facilities, personal workstations require a means for communicating with other nearby workstations and with servers that provide shared facilities such as file storage, printing, and long-haul communication. When the Alto design was started, we realized that such a facility would be needed, but did not understand its requirements well enough to begin a design. During late 1972 and early 1973, a number of alternatives were considered, ranging from star-connected serial links operating at a few hundred thousand bits per second to a parallel bus scheme operating at several million bytes per second. The need for bulk file transfers ruled out the low bandwidth of the first approach, and the complexity of the required cabling made the parallel bus unattractive. Coaxial cable, connected with standard cable television components, was tentatively selected as the transmission medium, since it would meet both the bandwidth and reliability requirements.

Several transmission methods were also considered, and a variant of the packet-based Aloha [1] radio system was selected. In pure Aloha, stations needing to transmit packets simply do so, and the resulting interference between stations reduces the channel capacity considerably. We realized that better performance was possible, since our cable Aloha stations could detect when their own transmissions were being interfered with, and abort them without transmitting a complete packet. We chose a baseband, as opposed to a carrier system because it is considerably simpler, and the extra bandwidth afforded by the latter scheme did not appear to be needed. These tentative decisions about the form that the network would take were made in late 1972, but little progress was made on an actual network design until Bob Metcalfe, who had joined CSL in mid-1972, and David Boggs, who came to SSL in March, 1973, began working on what was to become the Ethernet [31].

3 Implementation

In November 1972, implementation of the Alto began. The design was completed in approximately two months, including an initial version of the microcode for an instruction set emulator and for the display and disk controllers. Two prototype machines were then built using wirewrap technology, and were in operation in April 1973. One reason for the short schedule was that we had developed a rapid prototyping facility as a part of the construction of the Maxc timesharing system during 1971 and 1972. We were also able to use the memory boards originally developed for the Maxc system in the Alto, which saved considerable engineering effort. The design of the processor, memory subsystem, and display controller was done by Chuck Thacker, while the disk controller and its microcode were designed and im-

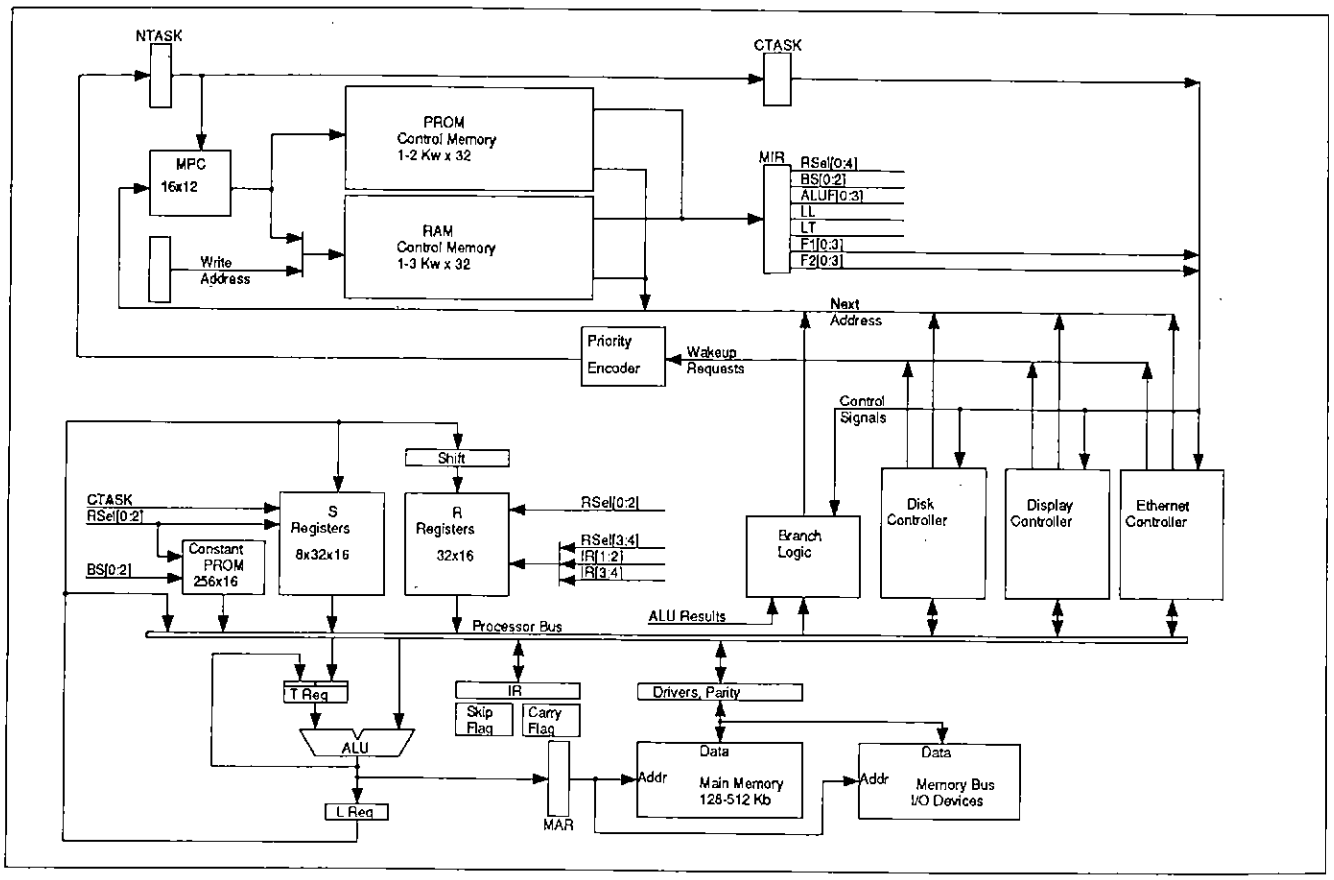


Figure 2: The Alto Microprocessor.

plemented by Ed McCreight. Larry Clark built the early systems and designed the package.

The Alto was a very simple machine by today's standards. The processor (Figure 2), is composed of three printed circuit boards containing about 200 small and medium-scale integrated circuits. Each of the input-output controllers occupies a single board containing approximately sixty integrated circuits. The processor is organized around a 16-bit bus connected to the main memory, an arithmetic unit, a number of high speed registers (R and S), and the input-output controllers. The transfer of data over the bus, the operations to be performed on the data, and the tests to be applied to it are controlled by a 32-bit microinstruction taken from PROM or RAM control store. Microinstructions are executed by a two stage pipeline which can start a new instruction every 170 nanoseconds. The processor is shared among sixteen fixed priority tasks. The NTASK and CTASK registers hold the number of the task currently in control of the processor. NTASK addresses the sixteen-element MPC RAM, which holds the task program counters. NTASK is loaded with the number of the highest priority wakeup request whenever the running task is willing to relinquish the processor. The only state associated with a task that is saved by the hardware is the task's program counter. Other machine resources are shared among tasks by programming convention. Normally, a task switch takes place with no overhead, unless it is necessary for the task giving up control to save and restore the L or T register.

The Alto main memory is synchronous with the processor, which starts all references by explicitly loading the memory address register (MAR). The memory can read or write a single 16-bit word in five machine cycles, or it can read a 32-bit doubleword in six cycles. The doubleword read was originally provided to support the display, which consumes two-thirds of the memory bandwidth even with this operation; it was also used very effectively by instruction set emulators for instruction fetching and to manipulate 32-bit quantities.

The Alto was not a high-performance machine, even by the standards of its time. Without the performance degradation caused by the input-output devices, it requires between one and three microseconds to execute a single emulated instruction. With the display running, these times are increased by a factor

of three. Until software was developed that required a great deal of computation for simple user actions (e.g., the Bravo text editor), the speed of the machine was adequate. Perhaps more important than the absolute speed is the fact that the performance of the Alto is *predictable*. It is very difficult to provide this characteristic in a timesharing system, and its lack can be very disconcerting to the user. The Alto cannot provide the peak performance of a timesharing machine, but it has the desirable property, pointed out by Jim Morris, that it doesn't run faster at night.

Once the design was complete, the microprocessor was simple enough that the hardware worked almost immediately. Debugging the microcode was more difficult, but was simplified considerably by an auxiliary writeable microstore built for the purpose. This device was connected to the control logic of the Alto under test. It replaced the PROM control store with RAM, and also added several bits to the microword. These additional bits were used to provide a breakpoint capability that made debugging much easier. The test unit was under control of a dedicated minicomputer that ran a microcode assembler and debugger. Using these tools, microcode debugging could be carried out as easily as debugging an assembly language program on a conventional machine.

The first of several microcoded instruction set emulators developed for the Alto was done for a virtual machine similar to the Data General Nova minicomputer. We had previously purchased a number of these systems, and had developed for them a compiler for BCPL [32], a predecessor of the popular C language. The main differences between the Alto instruction set (Figure 3) and that of the Nova were that the size of the Alto's address space was twice that of the Nova, and a number of instructions were added to support the Alto's input-output and interrupt system and to optimize BCPL procedure calls.

The resulting instruction set was not compatible with that of the Nova, but it was sufficiently similar that modifying the compiler was straightforward. Most of the early software for the Alto was written in BCPL. Only a small amount of assembly code was ever written for the machine. The microprocessor hardware contained a small amount of logic to enhance the performance of the BCPL emulator. This logic included a register to hold the instruction being executed and a method to address the registers used for the emulated machine's accumulators from its fields. This logic was not used by later emulators, and it probably would have been better to have used the same amount of hardware to provide functions with more general utility.

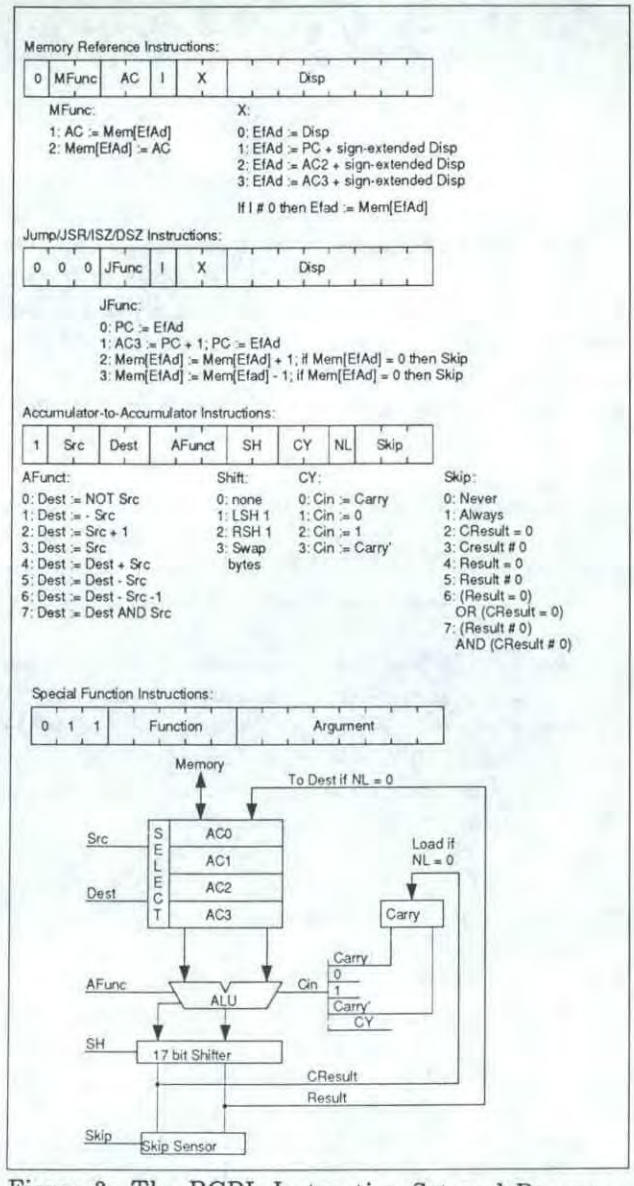


Figure 3: The BCPL Instruction Set and Processor Model

Emulators for several other instruction sets, including Smalltalk [18,16], Lisp [8,7], and Mesa [15] were written for the Alto. All were based on the idea of encoding the instructions as *bytecodes*. This arrangement allows instruction decoding to be done by a single 8-bit dispatch, rather than the several dispatches required to decode a BCPL instruction. The performance gained in this way more than offsets the lack of specialized decoding hardware. Most of the software written for the Alto after 1977 used Mesa, which was the choice in CSL, or Smalltalk, which was the language of choice in LRG. The Alto Lisp system was unsuccessful, primarily due to the lack of sufficient memory.

Input-output devices can be connected to the Alto in several ways. High performance controllers that make use of the microprocessor or require a high bandwidth connection to the memory are connected directly to the processor bus. The display, disk, and Ethernet controllers are examples of devices in this class. Lower performance devices are attached to the memory bus and addressed as locations at the end of memory, as in the PDP-11. The keyboard and keyset are connected in this way. Finally, a parallel port is provided for low performance devices outside the cabinet. This port was used to connect a variety of devices, including impact printers, a PROM programmer, and X-Y digitizing tablet, and a cassette tape deck.

The hardware controllers for the disk, the display, and the Ethernet are similar. They contain data buffering, logic to drive and receive control lines required by the device, and a small amount of device-specific timing and control logic. Most of the complexity associated with a controller is contained in its microcode. The display, for example, makes use of three of the Alto's sixteen microtasks: one is awakened during the display's vertical synchronization period, one is awakened during horizontal synchronization, and one is responsible for transferring raster data from main memory to the controller data buffer. The controller hardware provides wakeup requests at the appropriate times, but the microcode is responsible for carrying out most of the work required to maintain the display.

This arrangement, in which a device controller has the full computational power of the processor at its disposal, allowed us to provide convenient logical interfaces between devices and the driver software that operated them. The display interprets a chain of control blocks in main memory, each of which defines the contents of a horizontal band on the display screen. Since areas of the screen containing only white space, such as the space between lines and paragraphs, are not required to have any underlying bitmap memory, this technique reduces the display storage required by the standard text editor from the 61 thousand bytes required to represent a full screen to approximately 50 thousand bytes, a substantial saving in a machine with only 128 thousand bytes of memory. Drawing programs that allow the user to manipulate full-screen images cannot take advantage of this economy, but such programs are considerably simpler and smaller than the editor, so efficient use of space is less important in these applications.

The disk controller also makes extensive use of the Alto microprocessor. Like the display, it executes a chain of control blocks from main memory. Each control block specifies a 512 byte page to be read or written. The controller is able to transfer consecutive physical sectors between the disk and memory, which represented unusually high performance for the time. The controller uses two of the Alto's microtasks, one of which is awakened every sector, and one which is responsible for data transfers within a sector.

The disk controller and the file system were designed concurrently by Ed McCreight and Butler Lampson, with reliability as a primary goal. The designers wanted to provide a system in which hardware or software errors would result in a minimum amount of information loss. An important innovation—the use of label blocks—contributed substantially to the system's reliability. Labels add a third record to the header and data records customarily contained in a disk sector. The label record contains a unique integer identifier for the file containing the sector, the sector's position in the file, the addresses of the previous and next pages of the file, and the number of valid bytes in the data record. The controller microcode checks the information in the label block before doing any operation on the data record. This check ensures that the disk is properly positioned, both physically and logically, before any access to the data is done.

The use of label blocks, combined with replication of directory information in the first page of every file, means that the directory, which is an ordinary file, can be reconstructed from the contents of the disk itself if it becomes scrambled. Similarly, if data pages are corrupted, it is possible to determine precisely the extent of the loss, and to preserve the balance of the file system. One of the earliest programs written for the Alto was the Scavenger, which verified the integrity of a file system and corrected inconsistencies. This program, the first version of which was written by Jim Morris, makes the loss of even a small portion of a file system an extremely rare event.

The original Alto prototypes did not contain Ethernet interfaces, but during 1973, substantial progress on both network hardware and protocols was made. The name "Ethernet" was first used in May 1973 by Bob Metcalfe. Metcalfe and David Boggs worked on the network facilities during the summer and fall, and the prototype machines were exchanging packets by the end of 1973. Although the Alto was the first machine equipped with an Ethernet interface, Metcalfe and Boggs went on to design controllers for all the PARC computers, including Data General Novas and DEC PDP-11s. The Ethernet transceiver, an analog device that connects the controller to the coaxial cable, was designed by Tat C. Lam. The original Ethernet was slower than the commercial version available today. Its transmission rate was 2.94 million bits per

second, half the rate of the Alto master clock, and it used Manchester encoding for the serial data. For collision detection, it relied on comparison between the actual signal on the coaxial cable and the signal the transmitting station was attempting to send, rather than the level monitoring done by the commercial version. The 3 million bit per second bandwidth of the network was of some concern initially, in that we were not sure that it would be sufficient for a large system. Subsequent measurements [33] of a large network revealed that loads in excess of ten percent of full capacity were rare. There might have been less excess capacity if diskless workstations that paged over the network had been employed, but the use of local disks made this unnecessary.

In April, 1973, the first Alto prototype was completed. It was able to run simple programs to exercise the disk and display. The first image to be displayed was the Sesame Street 'Cookie Monster', which had been carefully digitized by a member of Kay's group.

During the balance of 1973, nine more prototype machines were built at PARC. During the summer, the prototypes had been tested sufficiently that we were willing to commit the design to printed circuit boards. At the same time, we realized that the lack of writeable control store would be a serious limitation in a machine intended for experimentation, and added one thousand words of instruction RAM to the original PROM control store. The original microcode was improved substantially, and a number of test programs for the hardware were written. Software development had begun, but there was not enough software available by the end of 1973 for the Alto to replace our timesharing system as the main computing facility for users.

One of the major strengths of the Alto in a research environment was that it could provide very high performance if the user were willing to accept the unpleasantness of microprogramming the processor directly. The first demonstration of such an application took place in the summer of 1973. Alan Kay, who was an accomplished organist, wanted a synthetic instrument with natural sound quality. He believed that this could be accomplished by recording and digitizing a real organ, and doing table lookup to provide a completely faithful replica of the waveshape. We decided to give him his wish, and purchased a two-manual organ keyboard (with pedals), a precision digital-to-analog converter, and a high fidelity amplifier and loudspeaker system. The keyboard and DAC were interfaced so that a task-specific function could read the keyboard as a bit vector, and load a value into the DAC. The organ simulator kept in main memory a table, consisting of the amplitudes at 256 points along a single cycle of the desired waveshape. Samples were generated for the DAC at a 25 kHz rate by taking points from this table at offsets which were inversely related to the frequency to be generated. The microprogram computed up to ten such samples every 40 microseconds, one for each key that was depressed. Samples were summed and the result was transmitted to the DAC. A variety of different effects could be generated by using different variations of the 'canonical cycle'. Although this application was fairly frivolous, it was an impressive demonstration of the real-time capability of the machine. Later, a serial line concentrator that connected up to sixty-four serial 300 baud lines to the Ethernet was built using similar techniques. The only extra hardware used in this device was a group of level converters and latches to allow the microprogram to read the value of the received data and store the data to be transmitted. All other processing was done by specialized microcode.

By late 1973, we were ready to produce a quantity of the machines for CSL and SSL. Although we had been able to build a small number of prototypes, the manufacture of the thirty systems we needed was beyond our capabilities. Fortunately, the company had established a custom systems manufacturing group in Los Angeles, and this group agreed to manufacture the Alto for us. The first machines were delivered between May and September, 1974. This was only slightly later than the first release of the basic Alto software, which took place in March, 1974. This early software consisted of the operating system [28], the BCPL compiler, and a primitive text editor. It allowed many of the researchers in CSL and SSL to begin doing a substantial amount of their work on the Alto, although the Maxc timesharing system was still used for electronic mail, file storage, and printing.

4 Servers

The most important components of a distributed computing system, after the workstations and the network that interconnects them, are the servers that provide shared facilities. We initially underestimated the importance of servers, assuming that the facilities provided by a set of workstations would be sufficient. We soon discovered that this was incorrect. Some functions, such as high-quality printing, are very expensive, and must be shared for economic reasons, while in other cases, sharing is used to provide communication between the users of the system. File storage is an example of the latter situation, although the low cost per byte of large disk files also provides economy of scale.

The first PARC server was EARS, a printing server named after its components: Ethernet, Alto, RCG (research character generator), and SLOT (scanning laser output terminal). This system was quite successful, in spite of the considerable obstacle to its acceptance presented by its bizarre name. It was the forerunner of the Xerox 9700 printer, which has been an extremely successful product.

The printing portion of EARS had been under development even before PARC was founded. Gary Starkweather, an optical engineer at the Xerox Webster Research Center, joined SSL in 1971. He brought with him a prototype printing engine consisting of a laser scanner attached to a standard Xerox copier. This device used a rotating polygon to scan an intensity-modulated laser beam across a standard xerographic drum, building up a raster image of the page being printed. During 1972, Ron Rider of SSL and Butler Lampson designed and implemented a character generator capable of printing high quality text in several fonts on Starkweather's engine, at a speed of one page per second.

A printer based on these components, driven by a Data General Nova minicomputer, was demonstrated in late 1973, but was never placed in service. Instead, Rider decided to build an Ethernet-based print server. He modified the character generator to allow it to be driven from an Alto, and wrote the necessary software to control the printer. The network-related portions of the server were written by Bob Metcalfe. EARS was the first major application for the Ethernet, and during its development, several experimental communications protocols [30] were tested and refined. EARS was placed in service in the Fall of 1974, and provided printing service to CSL and SSL until it was replaced by the Dover printer in 1977.

The final component needed for a complete distributed computing system was a file server. The hardware basis for several experimental file servers was a high-performance disk controller designed by Roger Bates in 1975. Like earlier Alto device controllers, it made use of the Alto's microprogrammed tasking for many of its functions, but it was considerably more complex than the earlier controllers because of the high bandwidth of the attached disks. Using this device, it was possible to connect as many as seven 300 million byte disks to a single Alto.

The first file server, Juniper, was to have provided page-level access to files, as well as atomic transactions. Planning for Juniper began in 1974, but actual programming did not begin until 1976. It was completed in 1977, but was never used extensively because of its poor performance.

The file server that was used most widely was the "Interim File Server", or IFS. This software was written in 1976 by Ed Taft and David Boggs, when it became clear that Juniper would not be ready as early as originally anticipated. It was an extension of the simple Alto file system, combined with the PUP (PARC Universal Packet) file transfer protocol [3]. It provided only bulk file storage, but it was completed rapidly and was reliable and efficient. However, as an "interim" system, it was a failure, since the IFS still provides the majority of the file storage in the Xerox internetwork.

The most complex server built using the Alto was the Dover printer, designed in 1976. Dover used a raster printing engine that was a descendant of the SLOT used in the EARS server, driven by a controller that relied heavily on the input-output processing capability of the Alto. Dover was a large project, involving several groups within the company. The design of the printing engine was done by the Special Projects Group (SPG) in Los Angeles, the group that manufactured the Alto. John Ellenby of CSL was responsible for management of the engine development. Software for the Dover was written by Dan Swinehart of SSL, and Bob Sproull of CSL. The development of the controller was done by Severo Ornstein of CSL, Bob Sproull, and Jim Leung of SPG, from a design by Butler Lampson.

The controller was considerably simpler than the earlier EARS character generator. Instead of using hardware to generate the bit stream for the printer in real time, the controller built up the image to be printed incrementally. Two buffer memories were used, each capable of holding sixteen of the printer's scan lines. While one of these buffers was being serialized and sent to the printer under hardware control, the other was being loaded with video information. The text to be printed and the bitmap representations for the fonts to be used were kept in Alto main memory, and written into the buffer by a high priority microcoded task. This process was then repeated for each of the roughly 250 bands that made up each page. Spooling of files received from the network was done by a BCPL program, which also sorted the contents of each page into bands in preparation for transmission to the controller. The resolution of the Dover was 384 pixels per inch, lower than the 500 pixels per inch of EARS, but still adequate for text and line graphics. The bandwidth requirements of the printer were high enough that the machine could not receive and spool files while printing, but its printing speed of one page per second was high enough that this was not a problem, since it could stop between pages to receive files. Incremental image generation also placed a limit on the complexity of pages that could

be printed. For the few images that exceeded the limit, another server that composed an entire page on disk and transmitted it to a slower printing engine was available.

The controller contained approximately 300 integrated circuits, making it about one eighth as complex as the character generator used in EARS. This simplification was made possible by the extensive use of the Alto's microprocessor to provide the low level control functions for the interface. Several dozen Dovers were built, and a number are still in operation.

5 Expansion

During 1975, the Alto was redesigned to improve its reliability and reduce its cost. The work was done by the Special Projects Group in Los Angeles that had been producing Altos for PARC, and was planned and supervised by John Ellenby of CSL. As part of this effort, all the boards and the package were redesigned. The memory system was reimplemented using 4 thousand bit RAM chips, and error correction was added. The resulting machine was much easier to build and service than the earlier system, and its cost was much lower—approximately twelve thousand dollars, rather than eighteen thousand dollars. The Los Angeles group had built a total of sixty of the original systems over a period of two years, most of which had been purchased by CSL and SSL. They were now able to produce the system in high volume. Over the next four years, approximately fifteen hundred Altos were built, of which approximately a thousand are still in use today. Most are used by individual engineers and scientists in a number of Xerox facilities, although many were configured as servers, and a few were used for marketing probes or donated to university computer science groups.

A final redesign done in 1979 replaced the memory with sixteen thousand bit RAM chips, and increased the amount of memory that could be attached to 512 thousand bytes. At the same time, the microcode store was changed from one thousand words of RAM and two thousand words of PROM to one thousand words of PROM plus three thousand words of RAM. By this time, other language emulators had almost totally superseded the original BCPL emulator. These emulators were usually loaded at bootstrap time or as part of starting a program that used a particular language. The additional RAM control store made it possible to spend less time minimizing the space required by the microcode and concentrate instead on its functionality and performance. The microcode was improved somewhat after the final redesign, but by 1980, most new development had shifted to the Alto's successors, described in the next section.

Although most of the software developed in CSL and SSL was distributed with the Alto, three application programs were primarily responsible for the machine's popularity with technical professionals in Xerox. The Bravo editor [22], designed by Charles Simonyi and Butler Lampson, and implemented and improved by Simonyi and others from 1974 through 1978, was the first and most important of these. Bravo was the first WYSIWYG (what you see is what you get) editor. It supported multiple fonts, and its high quality output could be printed on one of the many Dover printers that were available throughout the company.

The second important application program was the Laurel mail system [4], written by Doug Brotz, Roy Levin, and Mike Schroeder in 1978. Electronic mail has a profound effect on communication within an organization, since it combines the permanence and precision of memos with the speed of the telephone. By 1980, the Xerox internetwork, composed of local Ethernets connected by telephone lines, had been expanded to most of the engineering and research sites within the company. Laurel was rapidly adopted by a large fraction of this community, and by 1983 there were over four thousand regular electronic mail users in Xerox.

The third popular application was a group of tools for digital logic design, including the SIL illustrator [35] written by Chuck Thacker in 1975, and a routing program for prototypes written by Ed McCreight. These programs also produced high quality documentation on the Dover printers. They increased the productivity of designers significantly, and are used by most of the electronic engineers in the company.

6 Successors

The Alto was only the first of several personal workstations built at Xerox. The Dorado [26,6,27] and the Dicentra were developed at PARC, and the Dolphin and the Dandelion were commercial systems designed in the Electronics Division and the System Development Division.

Dorado is the largest hardware engineering project ever undertaken by the Computer Science Laboratory. It was difficult to think of the Dorado as a personal machine, since it consumed 2500 watts of power, was the size of a refrigerator, and required 2000 cubic feet of cooling air per minute (while producing a noise level that has been compared to that

of a 747 taking off).

It was *used* as a personal machine, however, and supplied computing power comparable to three VAX-11/780s. This may seem profligate, but it was consistent with the view that the CSL hardware base should be equivalent to that which would be commercially available and affordable in five to ten years. With hardware that is not limiting, it is possible to explore ways of using computers that are considerably ahead of current practice.

The Dorado project was started in CSL during 1975. It was moved to the System Development Division in 1976, but returned to CSL in 1977 when it became clear that the machine's high cost would not meet SDD's needs. The initial design was completed in late 1978, and two prototypes were built. A redesign, completed in 1979, was then done to simplify the machine. Manufacture of the machine started in 1980 in a small production facility that had been established for the purpose. By 1982, approximately thirty systems had been built, and the Dorado had replaced the Alto as the principal computing vehicle in CSL and SSL.

The Dorado achieved its high performance through its aggressive technology and a great deal of attention to detail on the part of its designers. It uses emitter-coupled logic (MECL 10K) with two to four nanosecond gate delays. The processor is microprogrammed, and like the Alto employs multitasking at the microcode level to operate input-output controllers. Unlike the Alto, it has virtual memory, an eight thousand byte cache, and a separate instruction fetch unit associated with the CPU. Up to 16 million bytes of main memory may be attached to the Dorado. The memory bandwidth available for input-output devices and to service cache misses is 66 million bytes per second. The processor executes microinstructions in a three-stage pipeline that can start a new instruction every sixty nanoseconds. The separate instruction fetch unit allows many instructions in the common emulators to execute in a single microinstruction.

In terms of man-years expended on a single project in CSL, the Dorado is second only to the Cedar programming environment [9,34], which its high performance made possible. The initial design was done by Butler Lampson and Chuck Thacker; the design was continued in SDD by Thacker, Brian Rosen, Don Charnley, and Tom Chang. When the project returned to PARC, it was partitioned into a number of subsystems: Ed McCreight and Severo Ornstein were the project managers; Butler Lampson was the technical leader of the project. The microprocessor was designed by Ken Pier, Roger Bates, and Ed Fiala. The instruction fetch unit was designed and implemented by Severo Ornstein, Gene McDaniel, and Will Crowther. The storage system was done by Doug Clark, Ed McCreight, and Ken Pier. A number of individuals produced the microcode for the machine, including Ed Taft, Peter Deutsch, Willie-Sue Haugeland, and Nori Suzuki.

The Dolphin was a much less ambitious successor to the Alto, designed in the Electronics Division of Xerox in 1977-79. In a sense, it was the successor of the Dorado, rather than the Alto, since it was done by the same group (Thacker, Charnley, Rosen, Chang) that had worked on the Dorado in SDD as well as a group in Los Angeles that included Jack Cameron, Howard Kakita, and Malcolm Thomson. Dolphin employed a number of ideas that had been incorporated into the Dorado, including virtual memory and a high-bandwidth input-output system. Its technology was not as aggressive as that of the Dorado—Schottky TTL, rather than ECL— and it was smaller and much less expensive. The Dolphin was used as the processor in the Xerox 5700 Electronic Printing System, and a version configured as a Lisp workstation became the 1100 Scientific Information Processor. Although PARC built approximately fifty Dolphins for internal use, and provided emulators for the Alto instruction set, Mesa, and Lisp, the machine was not popular. It had a higher resolution display, more memory, and a larger disk than the Alto, but it was only about twice as fast. Dolphin became available slightly before the Dorado, but the performance of the latter machine made it much more attractive, particularly for a research environment.

The Dandelion [19], known commercially as the Star 8010 workstation, was implemented in 1979 and 1980 by a group in SDD consisting of Bob Belleville, Robert Garner, and Ron Crane. Dandelion was based on a paper design called Wildflower done by Butler Lampson and Roy Levin of CSL. It was intended to have high performance and extremely low cost, but limited configuration flexibility. The Dandelion CPU used 2901 bit-slice processors, and employed a fixed time-slice form of multitasking that was quite different from that of the Alto. Dandelion was the first of the Alto's descendants that did not provide an emulation mode in which Alto software could be run. It was programmed exclusively in Mesa and the extended Mesa developed for the Cedar system. Dicentra, built by David Boggs and Hal Murray in 1982, was a variant of the Dandelion which included a Multibus rather than a proprietary bus for the attachment of input-output devices and memory. It provided a low-cost way to obtain a Mesa-compatible processor to which industry standard peripheral controllers and devices could be attached.

In addition to its direct descendants at Xerox, the Alto has inspired a number of similar systems from other commercial vendors. The Apple Lisa and Macintosh are perhaps the most familiar of these; Table 3 in Lampson [24] lists several others.

7 Conclusion

The Alto is small and slow by today's standards. The four generations of memory and microprocessor development that have passed since 1972 have made it straightforward to build low cost personal workstations with a hundred times the memory capacity and ten times the speed of the Alto. It seems likely that progress in semiconductors will continue at its present rate for perhaps another decade before fundamental physical limits are reached, so much more powerful systems are inevitable.

Higher bandwidth networks have also become much easier to engineer with the advent of fiber optics. However, experience with the Ethernet indicates that even with very high performance machines such as the Dorado, network bandwidth is not the limiting factor in overall system performance.

A surprising fact that has emerged from the work on the Alto and its successors is that the amount of software required to support interactive user interfaces is much greater than originally anticipated. Invariably, the complexity of the software is much greater than that of the hardware on which it runs. Except in a few applications in which the users are experts (e.g., programmers using programming environments), it has not yet become possible to provide the kind of symbiotic relationship between computer and human envisioned by Licklider in 1960. Advances in programming technology, as well as better hardware, will be required to achieve the kind of system he described. Distributed personal computing systems will help bring about these advances by providing more productive and efficient computing environments for developers.

References

[1] N. Abramson. The ALOHA System. In *Proc. AFIPS FJCC*, pages 281–285, 1970.

[2] D.G. Bobrow et al. Tenex: A paged time-sharing system for the PDP-10. *Communications of the ACM*, 15(3):135–143, March 1972.

[3] D.R. Boggs et al. Pup: An internetwork architecture. *IEEE Trans. Comm.*, 28(4):612–624, April 1980.

[4] D.K. Brotz. *Laurel Manual.* Technical Report CSL-81-6, Xerox Palo Alto Research Center, 1981.

[5] S. Card et al. Evaluation of mouse, rate-controlled isometric joystick, step keys and text keys for text selection on a CRT. *Ergonomics*, 21(8):601–613, August 1978.

[6] Douglas W. Clark, B.W. Lampson, and Kenneth A. Pier. The memory system of a high-performance personal computer. In *The Dorado: A High-Performance Personal Computer—Three Papers, CSL-81-1*, pages 51–80, Xerox Palo Alto Research Center, 1981.

[7] L.P. Deutsch. Experience with a microprogrammed Interlisp system. *IEEE Transactions on Computers*, C-28(10), October 1979.

[8] L.P. Deutsch. A Lisp machine with very compact programs. In *Proc. 3rd IJCAI*, Stanford, 1973.

[9] L.P. Deutsch and E.A. Taft. *Requirements for an experimental programming environment.* Technical Report CSL-80-10, Xerox Palo Alto Research Center, June 1980.

[10] D.C. Engelbart. The augmented knowledge workshop. In *Proc. ACM Conf. on History of Personal Workstations*, January 1986.

[11] D.C. Engelbart. A conceptual framework for the augmentation of man's intellect. In Howerton and Weeks, editors, *Vistas in Information Handling, volume 1*, pages 1–29, Spartan Books, Washington, 1963.

[12] D.C. Engelbart and W.K English. A research center for augmenting human intellect. In *Proc. AFIPS Conf.*, pages 395–410, 1968.

[13] E.R. Fiala. The MAXC systems. *IEEE Computer*, 11(5):57–67, May 1978.

[14] J.W. Forgie. The Lincoln TX-2 input-output system. In *Proc. Western Joint Computer Conf.*, pages 156–160, February 1957.

[15] C.M. Geschke et al. Early experience with Mesa. *Communications of the ACM*, 20(8):540–553, August 1977.

[16] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[17] D. Ingalls. The Smalltalk graphics kernel. *Byte*, 6(8):168–194, August 1981.

[18] D.H. Ingalls. The Smalltalk-76 programming system: Design and implementation. In *Proc. 5th ACM Symp. Principles of Prog. Lang.*, pages 9–16, January 1978.

[19] R.K. Johnsson and J.D. Wick. An overview of the Mesa processor architecture. *ACM Sigplan Notices*, 17(4):20–29, April 1982.

[20] A.C. Kay. Microelectronics and the personal computer. *Scientific American*, 237(3):236–245, September 1977.

[21] A.C. Kay. *The Reactive Engine.* PhD thesis, University of Utah, 1969.

[22] B.W. Lampson, editor. *Alto User's Handbook.* Xerox Palo Alto Research Center, 1976.

[23] B.W. Lampson. Guest editorial. *Software–Practice and Experience*, 2:195–196, 1972.

[24] B.W. Lampson. Personal distributed computing: The Alto and Ethernet software. In *ACM Conf History of Personal Workstations*, January 1986.

[25] B.W. Lampson et al. A user machine in a time-sharing system. *Proc. IEEE*, 54(12):1744–1766, December 1966.

[26] B.W. Lampson, Gene A. McDaniel, and Severo M. Ornstein. An instruction fetch unit for a high-performance personal computer. In *The Dorado: A High-Performance Personal Computer—Three Papers, CSL-81-1*, pages 21–50, Xerox Palo Alto Research Center, 1981.

[27] B.W. Lampson and K.A. Pier. A processor for a high-performance personal computer. In *Proc. 7th Symp. Computer Arch.*, pages 146–160, ACM Sigarch/IEEE, May 1980.

[28] B.W. Lampson and R.F. Sproull. An open operating system for a single-user machine. *ACM Operating Sys. Rev.*, 13(5), November 1979.

[29] J. Licklider. Man-computer symbiosis. *IRE Trans. Human Factors in Electronics*, HFE-1:4–11, March 1960.

[30] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[31] R.M. Metcalfe, D.R. Boggs, C.P. Thacker, and B.W. Lampson. U.S. Patent 4,063,220: Multipoint Data Communication System With Collision Detection. December 1977.

[32] M. Richards. BCPL: A tool for compiler writing and system programming. In *AFIPS Conf. Proc.*, pages 557–566, 1969.

[33] J.F. Shoch and J.A. Hupp. Measured performance of an Ethernet local network. *Communications of the ACM*, 23(12):711–721, December 1980.

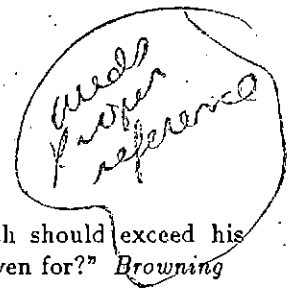
[34] W. Teitelman. A tour through Cedar. *IEEE Software*, 1(4), April 1984.

[35] C.P. Thacker. SIL—a simple illustrator for cad. In S. Chang, editor, *Fundamentals Handbook of Electrical Computer Engineering, Volume 3*, pages 477–489, Wiley, 1983.

[36] C.P. Thacker et al. Alto: A personal computer. In Siewiorek et al., editors, *Computer Structures: Principles and Examples*, chapter 33, McGraw-Hill, 1982. Also CSL-79-11, Xerox Palo Alto Research Center (1979).

Personal Distributed Computing: The Alto and Ethernet Software

Butler W. Lampson
Systems Research Center
Digital Equipment Corp.
130 Lytton Avenue
Palo Alto, CA 94301



Abstract

The personal distributed computing system based on the Alto and the Ethernet was a major effort to make computers help people to think and communicate. The paper describes the complex and diverse collection of software that was built to pursue this goal, ranging from operating systems, programming environments, and communications software to printing and file servers, user interfaces, and applications such as editors, illustrators, and mail systems.

1 Introduction

A substantial computing system based on the Alto [58] was developed at the Xerox Palo Alto Research Center between 1973 and 1978, and considerably extended between 1978 and 1983. The hardware base for this system is simple and uniform, if somewhat unconventional. The software, which gives the system its visible form and knits together its many parts, is by contrast complex, variegated, even ramified. It is convenient to call the whole complex of hardware and software "the Alto system." This paper describes the major software components of the Alto system. It also tries to explain why the system came out as it did, by tracing the ideas that influenced it, the way these ideas evolved into the concept of personal distributed computing, and the nature of the organization that built the Alto. A companion paper by Chuck Thacker [56] describes the hardware.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0101 $00.75

1.1 Themes

"But a man's reach should exceed his grasp, or what's a heaven for?" *Browning*

The Alto system grew from a vision of the possibilities inherent in computing: that computers can be used as tools to help people think and communicate [39]. This vision began with Licklider's dream of man-computer symbiosis [32]. When he became head of the Information Processing Techniques Office at ARPA, Licklider pursued the dream by funding the development of computer time-sharing. His successors, Ivan Sutherland, Robert Taylor, and Larry Roberts, continued this work throughout the 1960's, and extended it to computer networks. Taylor went on to manage the laboratory in which most of the Alto system was built, and people who worked on these ARPA projects formed its core. Their collective experience of time-sharing and networks became a major influence on the Alto design.

Another important influence, also funded by ARPA, was Doug Engelbart. He built a revolutionary system called NLS, a prototype of the electronic office. In NLS all the written material handled by a group of people is held in the computer and instantly accessible on a display which looks something like an ordinary sheet of paper. He demonstrated this system at the 1968 Fall Joint Computer Conference in San Francisco [12,13]. NLS was too expensive to be practical then, but it made a profound impression on many of the people who later developed the Alto.

Yet another ARPA project that had a strong influence on the Alto was Alan Kay's Flex machine, also called the Reactive Engine [21]. Kay was pursuing a different path to Licklider's man-computer symbiosis: the computer's ability to simulate or model any system, any possible world, whose behavior can be precisely defined. And he wanted his machine to be small, cheap, and easy for non-professionals to use. Like Engelbart, he attached great importance

to a high-quality, rapidly-changing display. He later coined the name "Dynabook" for the tool he envisioned, to capture its dynamic quality, its ubiquity, and its comfortable fit with people [22].

The needs of the Xerox corporation also influenced the Alto, since Xerox provided the money that paid for it. In 1970 Xerox started a research center in Palo Alto called PARC to develop the "architecture of information" and establish the technical foundation for electronic office systems which could become products in the 1980's. It seemed likely that copiers would no longer be a high-growth business by that time, and that electronics would begin to have a major effect on office systems, the firm's major business. Xerox was a large and prosperous company with a strong commitment to basic research and a clear need for new technology in this area. To the computer people at PARC, this looked like a favorable environment in which to take the next step toward making computers into effective tools for thinking.

The electronic office and the Dynabook, then, were the two threads that led to the Alto system. The idea of the electronic office is to do all the work of an office using electronic media:

- capturing information,
- viewing it,
- storing and retrieving it,
- communicating it to others, and
- processing it.

The idea of the Dynabook is to make the computer a personal and dynamic medium for handling information, which can model the world and make the model visible.

Of course the Alto system is not an electronic office, or a Dynabook; these were ideals to draw us on, not milestones to be achieved (or millstones either). In the course of developing the Alto we evolved from these ideals to a new style of computing, which is the preferred style in the 1980's just as time-sharing was the preferred style of the 1970's; witness the conference for which this paper was prepared. For this style there is no generally accepted name, but we shall call it "personal distributed computing." Why these words?

The Alto system is *personal* because it is under the control of a person and serves his needs. Its performance is predictable. It is reliable and available. It is not too hard to use.

And it is fast enough. In 1975 it was hard for people to believe that an entire computer is required to meet the needs of one person. The prevailing attitude was that machines are fast and people are slow; hence the merits of time-sharing, which allows one fast machine to serve many of the slow people. And indeed time-sharing, with response times measured in seconds, is an advance over a system which responds in hours. But this relationship holds only when the people are required to play on the machine's terms, seeing information presented slowly and inconveniently, with only the clumsiest control over its form or content. When the machine is required to play the game on the human's terms, presenting a pageful of attractively (or even legibly) formatted text, graphs, or pictures in the fraction of a second in which the human can pick out a significant pattern, it is the other way around: people are fast, and machines are slow. Indeed, it is beyond the current state of the art for a machine to keep up with a person. But the Alto system is a step in this direction.

So a personal system should present and accept information in a form convenient for a person. People are accustomed to dealing with ink on paper; the computer should simulate this as well as possible. It should also produce and accept voice and other sounds. Of these requirements, we judged most important the ability for the machine to present images, and for its user to point at places in the image. The Alto can do this quite well for a single $8.5'' \times 11''$ sheet of paper with black ink; it puts no restrictions on the form of the images. It cannot read images, nor can it handle sound; other people's experience suggest that these abilities, while valuable, are much less important.

The Alto system is *distributed* because everything in the real world is distributed, unless it is quite small. Also, it is implicit in our goals that the computer is quintessentially a communication device. If it is also to be personal, then it must be part of a distributed system, exchanging information with other personal computers. Finally, many things need to be shared, not just information, but expensive physical objects like printers and tape drives.

Finally, the Alto system is a *computing* system, adaptable by programming to a wide and unpredictable variety of tasks. It is big and fast enough that fitting the programs into the machine is not too hard (though the Alto's address space and memory size limitations were its most serious deficiency). And programming is possible at every level from the microcode to the user of an application. It is interesting, though, that user programming plays a much smaller role in the Alto system than in Unix [23] or Emacs [46]. Users *interact* with the system, nearly to the exclusion of *programming* it. In retrospect it seems

that we adopted this style because interacting with the Alto was so new and enthralling, and that we went too far in excluding systematic programming facilities like the Unix shell or M-Lisp in Emacs. The same mistake was made in early time-sharing systems such as CTSS [8] and the SDS 940 [26], whose builders were enthralled by a cruder kind of interaction, so we should have known better.

1.2 Schemes

"Systems resemble the organizations that produce them." *Conway*

The Alto system was affected not only by the ideas its builders had about what kind of system to build, but also by their ideas about how to do computer systems research. In particular, we thought that it is important to predict the evolution of hardware technology, and start working with a new kind of system five to ten years before it becomes feasible as a commercial product. If the Alto had been an article of commerce in 1974, it would have cost $40,000 and would have had few buyers. In 1984 a personal computer with four times the memory and disk storage, and a display half the size, could be bought for about $3500. But there is still very little software for that computer that truly exploits the potential of personal distributed computing, and much of what does exist is derived from the Alto system. So we built the most capable system we could afford within our generous research budget, confident that it could be sold at an affordable price before we could figure out what to do with it. This policy was continued in the late 1970's, when we built the Dorado, a machine with about ten times the computing power of an Alto. Affordable Dorados are not yet available, but they surely will be by the late 1980's. Our insistence on working with tomorrow's hardware accounts for many of the differences between the Alto system and the early personal computers that were coming into existence at the same time.

Another important idea was that effective computer system design is much more likely if the designers use their own systems. This rule imposes some constraints on what can be built. For instance, we had little success with database research, because it was hard to find significant applications for databases in our laboratory. The System R project [17] shows that excellent work can be done by ignoring this rule, but we followed it fairly religiously.

A third principle was that there should not be a grand plan for the system. For the most part, the various parts of the Alto system were developed independently, albeit by people who worked together every day. There is no uniform design of user interfaces or of system-wide data abstractions such as structured documents or images. Most programs take over the whole machine, leaving no room for doing anything else at the same time, or indeed for communicating with any other program except through the file system. Several different language environments are available, and programs written in one language cannot call programs in another. This lack of integration is the result of two considerations. First, many of the Alto's capabilities were new, and we felt that unfettered experimentation was the best way to explore their uses. Second, the Alto has strictly limited memory and address space. Many applications are hard enough to squeeze into the whole machine, and could not have been written if they had to share the hardware resources. Nor is swapping a solution to the shortage of memory: the Alto's disk is slow, and even fast swapping is inconsistent with the goal of fast and highly predictable interaction between the user and the machine.

There was some effort to plan the development of the Alto system sufficiently to ensure that the basic facilities needed for daily work would be available. Also, there was a library of packages for performing basic functions like command line parsing. Documentation for the various programs and subroutine packages was maintained in a uniform way and collected into a manual of several hundred pages. But the main force for consistency in the Alto system was the informal interaction of its builders.

The outstanding exception to these observations is the Smalltalk system, which was built by a tightly-knit group which spent a lot of effort developing a consistent style, both for programming and for the user interface. Smalltalk also has a software-implemented virtual memory scheme which considerably relaxes the storage limitations of the Alto. The result is a far more coherent and well-integrated world than can be found in the rest of the Alto system, to the point that several of the Alto's successors modelled their user interfaces on Smalltalk. The price paid for this success is that many Smalltalk applications are too slow for daily use. Most Smalltalk users wrote their papers and read their mail using other Alto software.

There was much greater consistency and integration of components in later systems that evolved from the Alto, such as Star and Cedar. This was possible because their designers had a lot of experience to draw on and 4-16 times as much main storage in which to keep programs and data.

The Alto system was built by two groups at PARC: the Computer Science Laboratory (CSL), run by

Robert Taylor and Jerome Elkind, and the Learning Research Group (LRG), run by Alan Kay. LRG built Smalltalk, and CSL built the hardware and the rest of the system, often in collaboration with individuals from the other computer-related laboratory at PARC, the Systems Science Laboratory. George Pake, as manager first of PARC and then of all Xerox research, left the researchers free to pursue their ideas without interference from the rest of Xerox.

The Star system, a Xerox product based on the Alto [43], was built by the System Development Division (SDD), run by David Liddle. The Dorado, a second-generation perconal computer [27] and Cedar, a second-generation research system based on the Alto [53], were built in CSL.

Table 1 enumerates the major contributors to the Alto and Cedar systems. Table 2 is a chronology of these systems.

1.3 What was left out

A number of ideas that might have been incorporated into the Alto system did not find a place there. In some cases we made deliberate decisions not to pursue them, usually because we couldn't see how to implement them on the Alto within an interactive application. Classical computer graphics, with viewports, transformations, clipping, 3-D projection and display lists, falls in this category; one implementation of a computer graphics package was done, but it didn't get used because it took up too much of the machine. High-quality typesetting (as in TeX [24]) is another; doing this interactively is still an unsolved problem. Linking together a large collection of documents so that the cross-references can be quickly and easily followed, as Engelbart did in NLS [13] is a third. Most of these ideas were incorporated into Cedar.

We also did very little with tightly-linked distributed computing. The 150 Altos at PARC could have been used as a multi-processor, with several of them working on parts of the same problem. But except for the Worm programs described in section 3, this was never tried. And, as we have already seen, the Alto system gave little attention to high-level programming by users.

Other things were tried, but without much success. We built two file servers that supported transactions, one low-level data base system, and several applications that used data bases. In spite of this work, by 1984 databases still did not play an important role in the Alto system or its successors. This appears to be because the data in our applications is not sufficiently well-structured to exploit the capabilities of a database system. Most Unix users apparently have had similar experiences.

We also tried to apply ideas from AI to office systems, with work on natural language understanding and on expert systems. But ten years of work did not yield systems based on these ideas that were useful enough to make up for their large size and slow speed. A full-scale Lisp system was built for the Alto, but the machine proved too small for it in spite of much effort; this system was later successful on larger machines.

One important idea which would have been a natural application for the Alto was simply overlooked: spreadsheets. Probably this happened because except for the annual budget planning and keeping track of purchase orders, there were no applications for spreadsheets in the research laboratory.

1.4 Overview

The Alto system software was begun in the middle of 1973. In October 1976 the Alto User's Handbook was published [25]. It describes a system that includes an operating system, a display-oriented editor, three illustrators, high-quality printing facilities, and shared file storage and electronic mail provided by local network communication with a time-sharing machine. There were also two programming languages and their associated environments. This was a complete personal distributed computing system; it met nearly all of the computing and information-handling needs of CSL and LRG.

The rest of the paper tells the story of the Alto system from the bottom up, except for the hardware, which is described elsewhere [56]. Section 2 covers *programming*: operating systems, programming languages, and environments. Section 3 presents the basic *communication* facilities: internet protocols, remote procedure calls, file transfer, and remote terminals. Section 4 discusses the *servers* which provide shared resources: long-term storage, printing, naming and mail transport.

The last two sections deal with the parts of the system that interact with users, which after all is the purpose of the whole enterprise. Section 5 considers the *user interfaces*: how the screen is organized into windows, how images are made, and how user input is handled. Section 6 describes the major *applications*, programs the user invokes to do some job with results that are useful outside the computer system. In the Alto system these are text editors, illustrators, electronic mail, and computer-aided design programs.

The conclusion reflects on the future of personal distributed computing and the nature of computer systems research.

Throughout I describe the Alto system in some detail, especially the parts of it that have not been described elsewhere. I also sketch its evolution into the Dorado system and the Xerox 8000 products (usually called Star). The "Dorado system" actually runs on three machines: Dolphin, Dorado and Dandelion. These have different microengines, but the same virtual memory structure, and all the major programming environments run on all three systems. They differ in performance and availability: the Dolphin, with about twice the power of the Alto and ten times the memory, first ran in 1978; the Dorado, ten times an Alto, in 1979; the Dandelion, three times an Alto, in 1980. Star runs on the Dandelion and was announced in 1981. A fourth member of the family, the Dicentra, was designed as a server and runs only the Mesa environment.

Readers may be surprised that so much of this paper deals with software which has little or nothing to do with the display, since 'screen flash' is the most obvious and most immediately attractive property of the Alto system. They should remember that the Alto system put equal emphasis on *personal* and on *distributed* computing, on *interacting* with the computer and on *communicating* through it. It also seems important to explain the programming environment that made it possible to develop the large quantity of software in the Alto system.

2 Programming environments

The Alto system is programmed in a variety of languages: BCPL, Mesa, and Smalltalk (see section 2.2). Each language has its own instruction set and its own operating system. The entire system has in common only the file system format on the disk and the network protocols. These were established by the BCPL operating system and did not change after 1976; they constitute the interface between the various programming environments.

2.1 Operating systems

The first Alto operating system [29], usually called the OS, is derived from Stoy and Strachey's OS6 [47]. It provides a disk file and directory system, a keyboard handler, a teletype simulator for the screen, a standard stream abstraction for input/output, a program loader, and a free storage allocator. There is no provision for multiple processes, virtual memory, or protection, although the first two were provided (several times, in fact) by software or microcode packages. The OS is written entirely in BCPL [40]. It was designed by Butler Lampson, and implemented by him, Gene McDaniel, Bob Sproull and David Boggs.

The distinctive features of the Alto OS are:

- its open design, which allows any part of the system to be replaced by a client program;
- the world-swap facility, which exchanges control between two programs, each of which uses essentially all of the machine;
- the file system, which can run the disk at full speed and uses distributed redundancy to obtain high reliability.

The OS is organized as a set of standard packages, one providing each of the functions mentioned (except for the stream abstraction, which is entirely a matter of programming convention). When it is initialized, all the packages are present and can be called by any program that is loaded. However, there is a *junta* procedure which can be used to remove any number of the pre-loaded packages and recover the space they occupy. Thus a program can take over nearly the whole machine. BCPL programs can include any subset of the standard packages to replace the ones removed by the junta. Alternatively, they can provide their own implementation of a function that has been removed by a junta, or do without it altogether. Thus the system is entirely *open*, offering services to its client programs but preempting none of the machine's resources. Other packages not normally loaded as part of the system, but available to be included in any user program, provide non-preemptive concurrent processes, internet datagrams and byte streams, program overlays, and other facilities that might have been part of the standard system.

At the base of the OS is a *world-swap* function which saves the entire state of the machine on a disk file and restores it from another file; this allows an arbitrary program to take control of the machine. The new program communicates with the old one only through the file system and a few bytes of state which are saved in memory across the world-swap. The world-swap takes about two seconds; it is used for bootstrapping, checkpointing, debugging (to switch between the program and the debugger), and switching back and forth between two major activities of a program.

The file system represents a file as a sequence of disk blocks linked by forward and backward pointers. By careful integration with the disk controller, it is able to transfer consecutive file blocks that are

contiguous on the disk at the full disk speed of 1 Mbit/second, while leaving time for non-trivial client computing; this performance allows world-swapping, program overlaying, and other sequential file transfers to be fast. In addition, each disk block contains the file identifier and block number in a *label* field of its header; this information is checked whenever the block is read or written. As a result, the disk addresses of file blocks can be treated as *hints* by the system, rather than being critical to its correct functioning and to the integrity of other files. If a disk address is wrong, the label check will detect the error, and various recovery mechanisms can be invoked. A Scavenger program, written by Jim Morris, takes about a minute to check or restore the consistency of an Alto file system; it can be routinely run by non-professional users. As a result of this conservative design, the file system has proved to be very reliable; loss of any information other than bits physically damaged on the disk is essentially unheard of. This reliability is achieved in spite of the fact that many programs besides the standard operating system have access to the disk.

The file system also runs on the 80 and 300 MByte disks that are interfaced to some Altos, and in this form is the basis of a successful file server (see section 4.2).

The Alto has a conventional hierarchical directory system which allows multiple versions of each file, with provisions for keeping a prespecified number of versions. The subdirectory and version facilities were added late, however, and did not enjoy widespread use. A directory is stored as an ordinary file which is processed by a variety of programs in addition to the file system.

A program called the Executive processes command lines and invokes other programs; it is much like the Unix Shell, but with far more primitive facilities for programming at the command level. From the point of view of the OS it is just another program, the one normally invoked when the machine is booted. From the point of view of a user it is the visible form of the OS.

Some of the design choices of the Alto OS would not be made in an operating system for a 1985 workstation; its much greater computing power and especially memory capacity make them unattractive. Separate address spaces simplify program development and concurrent execution of separate applications; virtual memory simlifies programming; a clean virtual machine makes it easier to port client programs to a new machine. But these comforts have a significant cost in machine resources: memory and response time. The open and unprotected character of the Alto OS were essential to the success of many Alto applications, both interactive programs like editors and illustrators, and realtime programs like the file, print, and gateway servers.

Distinct operating systems evolved for other programming environments. The Alto Lisp and Smalltalk operating systems were built using many of the Alto OS packages. Over time, these packages were replaced by new code native to the Lisp and Smalltalk environments. Mesa had its own operating system on the Alto from the start, but it was modelled closely on the OS.

For the Xerox 8000 products, the Alto OS was replaced by a considerably bigger system called Pilot, which supports virtual memory, multiple processes, and a more elaborate file system [38]. Pilot was also used in the Cedar system, but was eventually supplanted by the Cedar nucleus, a much simpler system quite similar in spirit to the Alto OS. Cedar also has CFS, the Cedar File System [41], which provides a network-wide file system to Cedar programs by caching files from a server on the workstation. In CFS modified files are written back only by explicit request. This arrangement supports about 20 Dorados from a single Alto-based file server, using about 30 MBytes of local disk on each Dorado for the cache. Almost any file server will do, since it need only support full file transfers.

2.2 Languages and environments

One of the distinctive features of the Alto as a programmer's system is the wide variety of programming environments it supports: BCPL, Smalltalk, Mesa, and Lisp. Each of these has its own language, which gives the environment its name, and its own instruction set, implemented by its own microcode *emulator*; the BCPL instruction set is always present, and there is a RAM with room for a thousand microinstructions, enough for one other emulator. Each also has its own compiler, loader, runtime system, debugger and library of useful subroutine packages. The ensuing subsections describe these parts for each of the Alto environments.

Distinct environments communicate only by world-swap, and through the file system. This isolation is not a good thing in itself, but it allows each system to stretch the resources of the machine in a different way, and thus to support effectively a particular programming style which would be impractical on such a small machine if all of the systems had to share a common instruction set and operating system.

2.2.1 BCPL

BCPL [40] is an implementation language quite similar to C (indeed, it is C's immediate ancestor). BCPL has a fairly portable compiler which Jim Curry ported to the Data General Nova, and thence to the Alto. Curry also wrote a loader for compiled code. It can produce an executable image or an overlay file which can be loaded and unloaded during execution; overlays are used in several large programs to make up for the limited address space. The Swat debugger, built by Jim Morris, is the other component of the BCPL programming environment; it understands BCPL's symbols, but is basically a machine-language debugger. The entire BCPL environment is very much like the C environment in Unix. Most Alto software was programmed in BCPL until 1977. By 1978 new programs were being written almost entirely in the other languages.

2.2.2 Mesa

Mesa [15] is an implementation language descended from Pascal. Its distinguishing features are:

- Strong type-checking, which applies even across separate compilation units.
- Modules, with the interface to a module defined separately from its implementation. Inter-module type-checking is based on the interface definitions; an implementation can be changed without affecting any clients [30].
- Cheap facilities for concurrent programming, well integrated into the language [28].
- A very efficient implementation, which uses an instruction set highly tuned to the characteristics of the client programs [20]. Compiled Mesa programs are typically half the size of similar C programs for the VAX, for example. The instructions are called byte-codes; many are a single byte long, and none are longer that three bytes. The byte-codes are interpreted by a microcoded emulator.
- A symbolic debugger well integrated with the source language and the type system, which allows breakpoints to be placed by pointing at the source code, and values to be printed in a form based on their type.

Mesa was begun in 1971 on a time-sharing computer by Jim Mitchell, Chuck Geschke and Ed Satterthwaite, ported to the Alto in 1975 with the assistance of Rich Johnsson and John Wick, and adopted by SDD in 1976 as the programming language for all SDD products. It continued to evolve there into a complete integrated programming environment [49]. A separate path of evolution in CSL led to the Cedar system described below.

The main goal of the Mesa research and development was to support the construction of large software systems and to execute them efficiently. This was accomplished very successfully: by 1982 there were several systems programmed in Mesa that contain more than a quarter of a million lines of code each, and many more that are 20-50 thousand lines long.

2.2.3 Smalltalk

The Smalltalk system is an integrated programming environment for the Alto, the first one to be built. It consists of a programming language, a debugger, an object-oriented virtual memory, an editor, screen management, and user interface facilities [22]. The latter are discussed in section 5.

The Smalltalk language is based on *objects* and *classes*; each object has a class, which is a collection of procedures that operate on the object. A *sub-class* inherits the procedures of an existing class, and generally adds new ones. The class of an object acts as its type, which is determined at runtime; when a procedure is applied to an object, the name of the procedure is looked up in the object's class to find the proper code to execute. Smalltalk source code is compiled into byte-codes similar to the ones used for Mesa, although much more powerful (and hence slower). The bytecodes are quite close to the source, so the compiler is small and fast; a typical procedure can be compiled and installed in a few seconds on an Alto, without disturbing the rest of the running system.

Objects are allocated and garbage-collected automatically. A class is itself an object, as are the source and compiled code for each procedure, and the stack frame for an executing procedure. So everything in the system can be accessed as an object and manipulated by Smalltalk programs within the system [19]. The entire system contains 100-200 classes, ranging from streams and collections to processes and files, rectangles and splines [16].

The dynamic typing, garbage collection, accessibility of every part of the system, and ease with which a procedure can be added or changed without any loss of state make Smalltalk similar to Lisp as a programming environment. The object-oriented programming style and the subclass structure, along with the careful design of the hierarchy of standard classes, give

Smalltalk its unique character. The system is fairly easy to learn, and very easy to use for building prototypes. Its limitations are relatively slow execution and (in the Alto implementations) modest memory capacity; these factors have prevented production system from being built in Smalltalk.

There have been three generations of the Smalltalk language and system: Smalltalk 72, Smalltalk 76, and Smalltalk 80; the description above refers to the latter two. The first two run on the Alto, the last on the Dorado and also on several VAX, 68000, and 80286 implementations.

2.2.4 Lisp

Several researchers in CSL used PDP-10 Interlisp [54,55] for their programming in the 1970s. Interlisp became part of the Alto system through software which allows the Alto to be used as a very capable graphics terminal [45] which supports multiple fonts and windows, along with the standard Alto raster graphics primitives (see section 5). The resulting Interlisp-D system [52] was the first to integrate graphics into a Lisp system.

A complete Interlisp system was built for the Alto by Peter Deutsch and Willie-Sue Haugeland [9,10]. It uses many of the BCPL OS packages for operating system functions, and the Lisp runtime is also written in BCPL. Lisp programs are compiled into byte-codes much like those used for Mesa and Smalltalk; they fall about half-way between those in the amount of work done by a single instruction. As with Mesa, programs are about twice as compact as when compiled for a conventional instruction set. The system uses an encoded form for list cells, which allows a cell to be represented in 32 bits most of the time, even though addresses are 24 bits.

Alto Lisp worked, and was able to run most of PDP-10 Interlisp, but it was too slow to be useful, mainly because of insufficient memory. It therefore did not play a role in the Alto system. The implementation was moved to the Dorado, however, where after considerable tuning it has been very successful [7].

2.2.5 Cedar

With the advent of the Dorado in 1979, we saw an opportunity to improve substantially on all of our existing programming systems. After considerable reflection on the desirable properties of a programming environment for personal computing [11], CSL decided in 1979 to build a new environment, based on Mesa, to exploit the capabilities of the Dorado. The goals were significantly better tools for program development, and a collection of high-quality packages implementing the major abstractions needed to write applications.

Cedar added several things to the program development system: garbage-collected storage, dynamic types, complete program access to the representation of the running program, a version control system, and uniform access to local and remote files. It also had a large assortment of generally useful packages: an integrated editor for structured documents, a powerful graphics package, a relational database system, remote procedure calls, user-programmable interpretation of the keyboard and mouse, and a screen manager based on icons and non-overlapping windows [51,53].

By early 1982 the Cedar system was usable; by mid-1984 it had grown to about 400,000 lines of code, ranging from the compiler and runtime support to an electronic mail system based on the Cedar editor and database system. It was possible to implement real-time servers such as a voice data storage system and a transactional file system, using the full facilities of Cedar.

Cedar was quite successful in overcoming the major limitations of the Alto programming environment. It also succeeded in providing good implementations for a number of widely-used high-level abstractions. However, an additional goal was to equal Lisp and Smalltalk in supporting rapid program changes and late binding, and here it was less successful. In spite of this limitation, it probably represents the state of the art in programming environments for workstations in 1984.

3 Communication

In order not to lose the ability of time-sharing systems to support sharing of information and physical resources in moving to personal computing, it was necessary to develop communication facilities far beyond the state of the art in 1973. The Ethernet provided the hardware base for this development. A great deal of work also went into the software.

The first generation of Alto communication software, called Pup for PARC Universal Packet [4], is based on ideas developed for the Arpanet in the late 1960's [33], with changes to correct known problems. Pup was originally designed by Bob Metcalfe, and implemented by him together with David Boggs and Ed Taft.

The model for communication was structured in four levels:

Level 0: Transport—various mechanisms for transporting a datagram from one machine to another: Ethernet, Arpanet, leased telephone lines, etc.

Level 1: Internet datagrams

Level 2: Interprocess communication primitives—byte system protocol (BSP), rendezvous/termination protocol, routing table protocol, etc.

Level 3: Data structuring conventions—file transfer protocol (FTP), remote terminal protocol (Telnet), mail transfer protocol (MTP), etc.

Internet datagrams are the common coin, decoupling higher levels from the details of the packet transport mechanism. Unlike other systems based on local networks, such as the Cambridge ring [36] or the Apollo Domain system [31], the Alto system takes no advantage of the special properties of the Ethernet. Instead, all clients of the communication facilities use internet datagrams, which present the same interface whether the data travels over the Ethernet, over a telephone line, or through a dozen different transport mechanisms. Pup thus treats a transport mechanism just as the Arpanet treats an Imp-to-Imp telephone line; the role of the Imp is played by a *gateway* which routes each datagram onto a transport mechanism which is expected to take it closer to its destination.

A major difference from the Arpanet is that Pup offers no guarantee that a datagram accepted by the network will be delivered. When congestion occurs, it is relieved by discarding datagrams. This design reflects the expectation that most datagrams will travel over lightly loaded local networks; when links become congested, clients of the network may get disproportionately degraded service.

The main level 2 mechanism is the byte stream protocol, which provides a reliable full-duplex byte stream connection between two processes. It turns out to be fairly hard to use this kind of connection, since it is complicated to set up and take down, and has problems when unusual conditions occur. As in the Arpanet, however, this was of little concern, since the main customers for it are the level 3 file transfer and remote terminal services. These are normally provided by programs called FTP and Chat, invoked by users or their command scripts from the Executive command line processor [25]. FTP was written by David Boggs, Chat by Bob Sproull. There is also an FTP library package that programs can call directly to transfer files; this is used by the Laurel mail-reading program, and (in an abbreviated version) by many applications that send files to the printer.

Pup byte streams have fairly good performance. They are capable of transferring about .3 MBits/second between two Altos, using datagrams with 512 data bytes plus a few dozen bytes of overhead. This is an order of magnitude better than the performance exhibited by a typical byte stream implementation on a comparable machine. The difference is due to the simplicity of the Pup protocols and the efficient implementation.

Other important network services in the Alto system were booting a machine and collecting reports from diagnostic programs. These used internet datagrams directly rather than a level 2 protocol, as did one rather successful application, the Woodstock File System [50] described in section 4. But with these few exceptions, users and applications relied on FTP and Chat for communication services.

The only truly distributed program built in the Alto system other than the Pup internet router, is the Worm [42], which searches for idle machines into which it can replicate itself. It isn't entirely clear why more attempts were not made to take advantage of the 150 Altos or 50 Dorados available at PARC. Part of the reason is probably the difficulty of building distributed programs; this was not alleviated by RPC until 1982 (see below). The lack of a true network-wide file system before 1984 was probably another factor.

The Xerox 8000 network products use an outgrowth of Pup called Xerox Network System or XNS [60]. XNS adds one important level 3 protocol, the Courier remote procedure call protocol designed by Jim White. Courier defines a standard way of representing a procedure call, with its argument passing and result returning, between two machines. In other words, it includes conventions for identifying the procedure to be called, encoding various types of argument (integers, strings, arrays, records, etc.), and reporting successful completion or an exceptional result. Courier uses XNS byte streams to transport its data.

At about the same time, as part of the development of Cedar, Andrew Birrell and Bruce Nelson in CSL developed a remote procedure call mechanism that uses internet datagrams directly [2]. They also built a *stub generator* that automatically constructs the necessary program to convert a local procedure call into a remote one, and a location mechanism based on Grapevine (see section 4.3) to link a caller to a remote procedure. Like Courier, Cedar RPC uses a Mesa interface as the basic building block for defining remote procedures. Unlike Courier, Cedar RPC is fast: two Dorados using Cedar RPC can execute a null remote procedure call in about a millisecond.

Earlier work by Nelson indicates that further optimization can probably reduce this to about 200 microseconds; currently Cedar RPC has no clients that are limited by the performance.

The goal of Cedar RPC was to make it possible to build distributed programs without being a communications wizard. A number of successful weekend projects have demonstrated that the goal was reached. An unexpected byproduct was independent implementations of the same RPC mechanism in Lisp, Smalltalk, and C for the 8088; all these clients found it easier to implement the Cedar RPC than to design a more specialized mechanism for their own needs.

4 Servers

The main use of distributed computing in the Alto system is to provide various shared services to the personal computers. This is done by means of *servers*, machines often equipped with special input-output devices and programmed to supply a particular service such as printing or file storage. Although there is no reason why a single machine could not provide several services, in fact nearly all the servers are Altos, and have no room for more than one server program.

4.1 Printing

Printing is the most complex and most interesting service in the Alto system. A considerable amount of research had to be done to build practical printers that can make high-quality hard copies of the arbitrary images the Alto screen can display. The end product of this work at PARC was a printing system that stores many thousands of typeset pages, and can print them at about 40 pages per minute. The quality is fairly close to a xerographic copy of a professionally typeset version. All the documentation of the Alto system and all the technical papers, reports and memos written in the computer research laboratories were stored in the system and printed on demand.

There are several aspects of a printing service. First, there must be a printing *interface*, a way to describe the sequence of pages to be printed, by specifying the image desired on each page as well as the number of copies and other details. Second, there must be a *spooler* that accepts documents to be printed and queues them. Third, there must be an *imager* that converts the image descriptions into the raster of bits on the page, and sends the bits to the printing hardware at the right speed.

The spooler is fairly straightforward: nearly all the printers include a disk, accept files using a standard file transfer protocol, and save them in the OS file system on the disk for printing. Since imaging consumes the entire machine, it is necessary to interrupt printing to accept a new file, but file transfers are fast, so this is not a problem. The interfaces and the imagers are worth describing in more detail.

The main point of a printing interface is to decouple creators of documents from printers, so that they can develop independently. Four printing interfaces were developed for the Alto and Dorado systems, each providing more powerful image description and greater independence from the details of printer and fonts. The first, devised for the XGP printer by Peter Deutsch, is rather similar to the interfaces used to control plotters and dot-matrix printers nearly fifteen years later: it consists of the ASCII characters to be printed, interspersed with control character sequences to change the font, start a new line or a new page, justify a line, set the margins, draw a vector, etc. The graphics capabilities are quite limited, and the printer is responsible for some of the formatting (e.g., keeping text within margins).

This interface, with its lack of distinction between formatting a text document and printing, proved quite unsatisfactory; there was constant demand to extend the formatting capability. All the later interfaces take no responsibility for formatting, but require everything to be specified completely in the document. Thus line and page breaks, the position of each line on the page, and all other details are determined before the document is sent to the printer. This clear separation of responsibilities proved to be crucial in building both good printers and good document formatters.

The second printing interface was designed by Ron Rider for use with the Ears printer, the first raster printer capable of high-quality printing. The graphics capability of the Ears printer is limited, and as a consequence the Ears interface can only specify the printing of rectangles; usually these are horizontal or vertical lines. It does, however, allow and indeed require the document to include the bitmaps for all the fonts to be used in printing it. Thus font management is entirely decoupled from printing in Ears, and made the responsibility of the document creator. The great strength of this interface is the complete control it provides over the positioning of every character on the page, and the absence of restrictions on the size or form of characters; arbitrary bitmap images can be used to define characters. Font libraries were developed for logic symbols so that logic drawings could be printed, and other programs generate new fonts automatically for drawing curves.

After several years of experience with Ears, the advent of two new printers stimulated the development

Pren Edit

of a new, printer-independent interface called Press, designed by William Newman and Bob Sproull. Its main features are a systematic imaging model, arbitrary graphics, and a systematic scheme for naming fonts rather than including them in the document, since font management proved to be too hard for document creators. Thus the printer is responsible for storing fonts, and for drawing lines and spline curves specified in the document. The device-independence of Press makes it possible to display Press documents as well as print them.

Press served well for about six years. In 1980, however, the development of general-purpose raster printing products motivated yet another interface design, this time called Interpress; it was done by Bob Sproull and Butler Lampson, with assistance from John Warnock [44]. The biggest innovation is the introduction of a stack-based programming language, which contains commands to draw a line or curve, show a character, etc., as well as to manipulate the stack and call procedures. The document is a program in this language, which is executed to generate the image. This design provides great power for describing complex images, without complicating the interface. Interpress also has extensive provisions to improve device-independence, based on experience in implementing Press for a variety of printers. It handles textures and color systematically.

Imagers evolved roughly in parallel with interfaces. The task of generating the 10-25 million bits required to define a full-page raster at 300-500 bits/inch is not simple, especially when the printer can generate a page a second, as most of those in the Alto system can. Furthermore, the fonts pose a significant data-management problem, since a single type-face requires about 30 KBytes of bitmap, and every size and style is a different type-face. Thus one serif, one sans-serif and one fixed-pitch font, in roman, italic, bold, and bold italic, and in 6, 7, 8, 9, 10, 12, 14, 18 and 24 point sizes, results in 108 type-faces; actually the system uses many more.

The first imager, written by Peter Deutsch, drove the Xerox Graphics Printer. This machine is slow (5 pages/minute) and low resolution (200 dots/inch); it is also asynchronous, so the imager can take as long as it likes to generate each line of the raster. Only crude imaging software was developed for it, together with a few fonts. Raster fonts were unheard of at the time, except in 5x7 and 7x9 resolution for terminals, or at very high resolution for expensive photo-typesetters. The XGP fonts were developed entirely manually, using an editor that allows the operator to turn individual dots in the roughly 20x20 matrix on and off. They had to be new designs, since it is impractical to faithfully copy a printer's font at such low resolution. These XGP fonts were later widely used in universities.

The second imager drives Ears, a 500 dot/inch, 1 page/second xerographic printer based on a Xerox 3600 copier engine and a raster output scanner developed at PARC by Gary Starkweather. Ears is controlled by an Alto, but there is a substantial amount of special-purpose hardware, about three times the size of the Alto itself, to store font bitmaps and generate a 25 MHz video signal to control the printer. Both the imager and the hardware were developed by Ron Rider, with design assistance from Butler Lampson. This machine revolutionized our attitude to printing. The image quality is as good as a xerographic copy of a book, and it can print 80 pages an hour for each member of a 40-person laboratory (of course we never used more than a fraction of its capacity). A second copy of Ears served as the prototype for the Xerox 9700 computer printer, a very successful product. Fonts for Ears were produced using the Fred spline font editor described in section 6.

As Alto systems spread, there was demand for a cheaper and more easily reproduced printing server. This required a new printing engine, new imaging hardware, and a new imager. The engine was the Dover, also based on the 3600 copier, but at 384 dots/inch; its development was managed by John Ellenby. Bob Sproull and Severo Ornstein developed the Orbit imaging hardware based on a design by Butler Lampson; this is about half the size of the Alto that drove it, a better balance. Sproull and Dan Swinehart wrote the Spruce imager to accept Press files and drive the Dover and Orbit. About 50 copies of this system were made and distributed widely within Xerox and elsewhere. It was cheap enough that every group could have its own printer. Spruce includes the font management required by Press, and is normally configured with an 80 MByte disk for font storage and spooling. Although Press can specify arbitrary graphics, Spruce handles only lines, and it can be overloaded by a page with too many lines or small characters. These limitations are the result of the fact that Orbit has no full-page image buffer; hence Spruce must keep up with the printer, and may be unable do to so if the image is too complex.

Another Press imager, confusingly also called Press, was built by Bob Sproull and Patrick Baudelaire to drive two slower printing engines, both at 384 dots/inch, and one able to print four colors. This imager constructs the entire 15 MBit raster on a disk, and then plays it out to the printer; this is possible because the engines are so much slower. It was the

first Press imager, and the only one able to handle arbitrary images, by virtue of its complete raster store. Among other things, Press produced some spectacular color halftones.

The first Interpress imager was developed by Bob Ayers of SDD for the 8044 printer, part of the Xerox 8000 network product family. Subsequently Interpress imagers have been built for several other engines.

4.2 Storage

Although not quite as exciting as printing, shared storage services are even more necessary to a community of computer users, since they enable members of the community to share information. No less than five file servers were developed for the Alto and Dorado systems.

For its first three years, the Alto system used CSL's mainframe [14], running the Tenex operating system [3], as its file server. A Pup version of the Arpanet file transfer program, written by Ed Taft, provides access to the Tenex file system. Both Alto and Tenex have both user and server ends of this program, so either machine can take the active role, and files can also be transferred between two Altos.

The second file server, called WFS and written by David Boggs [50], is near the opposite extreme of possible designs. It uses a connectionless single-datagram request-response protocol, transfers one page of 512 bytes at a time, and implements only files; directories are the responsibility of the client. There is a very simple locking mechanism. This extreme simplicity allowed WFS to be implemented in two months on a Data General Nova; it was later ported to the Alto. Of course, only clients that provided their own naming for data were possible, since WFS had none. These included an experimental office system called Woodstock, the Officetalk forms-handling system, Smalltalk, and a telephone directory system.

As the Alto system grew, the file service load on Tenex became too great. Since the Alto has a disk controller for 300 MByte disks, it is possible to configure an Alto with several GBytes of storage. David Boggs and Ed Taft assembled existing Alto packages for files (from the OS), file transfer (from FTP), and directories (a B-Tree package written by Ed McCreight) to form the Interim File System (IFS). A considerable amount of tuning was needed to fit the whole thing into a 128 KByte Alto memory, and a new Scavenger had to be written to handle large disks, as well as an incremental backup system. IFS served as the main file server throughout the Alto system for at least seven years and dozens of IFS servers were installed.

IFS supports multiple versions of files, write locking, and a clumsy form of sub-directories. Over time it acquired single-packet protocols to report the existence of a file to support CFS (section 2.1), a WFS-style file server, an interum mail server, and access to Grapevine (section 4.3) for access control.

Concurrent with the development of WFS and IFS was a research project to build a random-access multi-machine file server that supports transactions and fine-grained locking. This became the Juniper system, designed and developed by Howard Sturgis, Jim Mitchell, Jim Morris, Jay Israel, and others [34]. It was the first server written in Mesa, the first to use an RPC-like protocol (section 3), and the first to attempt multi-machine computation. Juniper was put in service in 1977, but performance on the Alto was marginal. Later it was moved to the Dorado. The goals proved to be too ambitious for this initial design, however, and it was never widely used.

Cedar includes a second-generation file server supporting transactions. Called Alpine, it was built by Mark Brown, Ed Taft, and Karen Kolling [6]. Alpine uses Cedar RPC and garbage-collected storage; it also supports multi-machine transactions and fine-grained locks, using newer algorithms that benefit from the experience of Juniper and System R [17]. It is widely used in Cedar, especially to store databases.

4.3 Naming and mail transport

The Alto system initially relied on Pup host names to locate machines, and on the Tenex mail system for electronic mail. These worked well until the system grew to hundreds of machines, when it began to break down. It was replaced by Grapevine, designed and implemented by Andrew Birrell, Roy Levin, Roger Needham and Mike Schroeder. Grapevine provides naming and mail transport services in a highly available way [1]. It also handles distribution lists and access control lists, both essential for a widespread community.

Grapevine was the second Mesa server. It was put into service in 1980 and has been highly successful; dozens of servers support a community of about two thousand machines and seven thousand registered users. All the machines in the Alto and Dorado systems use it for password checking and access control, as well as for resource location and mail transport. It is also used to register exporters of RPC interfaces such as Alpine file service. The data base is replicated; in other words, each item stored on several servers; as a result it is so reliable that it can be depended on by all the other components of the

distributed system.

5 User interfaces

Perhaps the most influential aspects of the Alto system have been its user interfaces. These of course depend critically on the fact that the screen can display a complex image, and the machine is fast enough to change the image rapidly in response to user actions. Exploiting these capabilities turned out to be a complex and subtle matter; more than ten years later there is still a lot to be learned. The ensuing description is organized around four problems, and the Alto system's various solutions to them:

- organizing the screen,
- handling user input,
- viewing a complex data structure, and
- integrating many applications.

A final sub-section describes the facilities for making images.

5.1 Windows

As soon as the limited display area of the screen is used to show more than one image, some scheme is needed for multiplexing it among competing demands. There are two basic methods:

- *switching*, or time-multiplexing—show one image at a time, and switch quickly among the images;
- *splitting*, or space-multiplexing—give each image some screen space of its own.

Various approaches were tried, usually combining the two methods. All of them organize the major images competing for space, typically text documents or pictures of some kind, into rectangular regions called *windows*. All allow the user some control over the position and size of the windows. The windows either *overlap* or they *tile* the screen, arranged in one or two columns and taking up the available space without overlapping. Thus overlapping is a combination of splitting and switching; when two windows overlap, some of the screen space is switched between them, since only the one on top can be seen at any instant. Where windows don't overlap, the screen is split. Tiling, by contrast, is a pure splitting scheme.

With overlapping, the size of one window can be chosen independently of the size or position of others, either by the system or by the user. With tiling, this is obviously not the case, since when one window grows, its neighbor must shrink. As a practical matter, this means that a tiled system does more of the screen layout automatically.

An overlapped system can have any number of windows on the screen and still show the topmost ones completely, perhaps at the cost of completely obscuring some others so that the user loses track of them. In a tiled system, as more windows appear, the average size must become smaller. In practice, four or five windows in each of one or two columns is the limit. Either system can handle more windows by providing a tiny representation, or *icon*, as an alternative to the full-sized window. This is another form of switching.

A minor image, usually a menu of some kind, either occupies a sub-region of a window (splitting), or is shown in a *pop-up* window that appears under the cursor (switching). The pop-up temporarily obscures whatever is underneath, but only for a short time, while a mouse button is down or while the user fills in a blank. Thus it has an entirely different feeling from a more static overlapped window.

Figures 1-3 are typical screen arrangements from three systems. Smalltalk (figure 1) uses overlapping windows without icons, and the position of a window is independent of its function (unless the user manually arranges the windows according to some rule). The small skinny window is a pop-up menu. Smalltalk was the first system to use overlapping windows and pop-up menus. The Bravo editor (figure 2) uses one column of tiled windows, with a control window at the top, a message window at the bottom, and a main window for each document being edited, which may be subdivided to look at different parts. Cedar (figure 3) uses two tiled columns and rows of icons at the bottom (which can be covered up). This window system is called Viewers; much of its design was derived from Star. The top line or two of a window is a menu. Cedar also allows the entire screen image, called a *desktop*, to be saved away and replaced by another one; this is switching on a large scale. Markup has a pop-up menu scheme like Smalltalk's, but considerably more elaborate (figure 4).

5.2 Input

User input comes from the keyboard, the mouse buttons, or the cursor position, which is controlled by the mouse. The crucial notion is *clicking* at an object on the screen, by putting the cursor near it and pushing a button. There are three mouse buttons, so three kinds of click are possible; sometimes multiple clicks within a short interval have different meanings; sometimes the meaning of a click is modified by shift

keys on the keyboard; sometimes moving the mouse with a button held down has a different meaning. An interface for novices will draw sparingly from this variety of mouse inputs, but one designed for experts may use all the variations.

If the screen object clicked is a *menu button*, this gives a *command* that makes something happen; otherwise the object is *selected*, in other words, designated as an operand for a command, or some point near the object is designated, such as an insertion point for text. The selection is made visible by underlining or reversing black and white; the insertion point is made visible as a blinking caret, I-beam, or whatever.

Many interfaces also have *scroll bars*, thin rectangles along one side of a window; clicking here scrolls the document being viewed in the window so that another part of it can be viewed. Often a portion of the scroll bar is highlighted to indicate the position and size of the region being viewed relative to the whole document. A *thumbing* option scrolls the view so that the highlight is where the mouse is; thus thumbing a quarter of the way down the bar scrolls a quarter of the way into the document.

Frequently commands are given by keystrokes as well as, or instead of menu clicking. Nearly always the commands work on the current selection, so the selection can be changed any number of times before a command is given without any effect on the state. An expert interface often has mouse actions which simultaneously make a selection and invoke a command, for instance to open an icon into a window by double-clicking on it, or to delete an object by selecting with a shift key depressed.

Most Alto system interfaces are *modeless*, or nearly so: any keystroke or mouse click has the same general meaning in any state, rather than radically different meanings in different states. For example, an editor is modeless if typing the "A" key always inserts an "A" at the current insertion point; an editor is not modeless if typing "A" sometimes does this, and sometimes appends a file. A modeless interface usually has postfix commands, in which the operands are specified by selections or by filling in blanks in a form before each command is given.

Many of the user input ideas described here were first tried in the Gypsy editor (see section 6); others originated in Smalltalk, yet others in the Markup or Sil drawing programs (also described in section 6). One of the main lessons learned from these and other experiments is the importance of subtle factors in the handling of user input. Apparently minor variations can make a system much easier or much more difficult to use. In Bravo and Cedar this observation led to schemes which allowed the expert to rearrange the interpretation of user input, attaching different meanings to the possible mouse actions and keystrokes, and redesigning the menus. This decoupling of input handling from the actions of an application has also been successful in EMACS [46].

5.3 Views

More subtle than windows, menus, and double-click selections, but more significant, is the variety of methods in the Alto system for presenting on the screen a meaningful view of an abstract structure, whether it is a formatted document, a logic drawing, the tree structure of classes in Smalltalk, a hierarchical file system or the records in a database. Such a view represents visually the structure and content of the data, or at least an important aspect of it. Equally important, it allows the user to designate part of the data and change it, by pointing at the view and giving commands. And these commands have immediate visual feedback, allowing the user to verify that the command had the intended effect and reinforcing a sense of direct contact with the data stored in the machine.

There are many ways to get this effect; all the methods that have been developed over the centuries for presenting information visually can be applied, along with others that depend on the interactiveness of the computer. A few examples must suffice here.

Perhaps the most familiar is the "what you see is what you get" editor. The formatted document appears on the screen, complete with fonts, subscripts, correct line breaks, and paragraph leading; page layout should appear as well, but this was too hard for the Alto. The formatted text can be selected, and the format as well as the content changed by commands which show their effect immediately. Figure 2 is an example from Bravo.

The Smalltalk browser shown in figure 1 is a visual representation of part of the tree of Smalltalk classes and procedures. The panels show successive levels of the tree from left to right; the highlighted node is the parent of all the nodes in the panel immediately to its right. Underneath the panels is the text content of the last highlighted node, which is a leaf of the tree. Again, sub-trees can be selected and manipulated, and the effect is displayed at once.

The list of message headers in figure 3 is another example. Each line shows essential properties of a message, but it also stands for the message; it can be selected, and the message can be deleted, moved or copied to another file, read, or answered. The same methods are used in Gypsy and Star to represent a

hierarchical file system, using the visual metaphor of files in a folder and folders in a file drawer. The icons on the screen in Star, Cedar, and more recent systems like the Apple Macintosh repeat the theme again.

The idea of views was first used in Sketchpad, which provides a graphical representation of an underlying structure of constraints [48], but it was obscured by the fact that Sketchpad was intended as a drawing system. In the Alto system it appeared first in Gypsy and Bravo (for formatted text and files), then in the Smalltalk browser, and later in nearly every user interface. It is still a challenge, however, to devise a good visual representation for an abstraction, and to make the pointing and editing operations natural and fast.

5.4 Integration

An integrated system has a consistent user interface, consistent data representations, and co-resident applications. What does this mean?

A consistent user interface is one in which, for example, there is a single 'delete' command which deletes the selected object, whether the object is a text string, a curve, a file, or a logic element. Every kind of object has a visual representation, and operations common to several objects are specified in the same way and have the same results on all the objects. Most computer systems are constructed by amalgamating several applications and lack any consistency between the applications.

A consistent data representation is one that allows two applications handling the same kind of data to accept each other's output. Thus a document editor can handle the drawing produced by an illustrater or a graph plotting program, and a compiler can read the output of a structured document editor, ignoring the structuring information that isn't relevant to the syntax of the programming language.

Two applications are co-resident if control can pass from one to the other quickly and smoothly, without loss of the program state or the display state. If a text document editor allows pictures in the document, can pass control to the picture editor, and the picture can be edited while it appears on the screen along with the text, then the text and picture editors are co-resident. In the same way a spreadsheet, a data base query system, or a filing system might be co-resident with editors or with each other.

Integration is difficult for several reasons. Designing consistent interfaces is difficult, both technically and organizationally. Consistent data representations require compromises between the different needs of different applications; most Alto programmers were uninterested in compromise. Co-residency requires common screen-handling facilities that meet the needs of both applications, and enough machine resources to keep the code and data for more than one program readily available.

In spite of these problems, integration was always a goal of many Alto system applications, but success was limited. Gypsy has excellent integration of text editing and filing. The window system of Smalltalk provides some integration; the later Star and Cedar systems continue this, and all three have some consistency in user interfaces. All three also have a common data representation for simple text, and make it easy to copy text from one application to another as pipes do in Unix. Star goes furthest in consistency of interfaces and representations, and also has co-residency of all its applications. A great deal of effort was devoted to the user interface design [43] to achieve these results. Unfortunately, it pays a considerable penalty in complexity and performance. In Cedar some applications, notably the Tioga structured document editor, can be easily called as subroutines from other applications.

5.5 Making images

The Alto user interface depends heavily on displaying images to the user. These are of two main kinds: text and pictures. The application programs that allow users to make different kinds of images are described in the next section. They all, however, use a few basic imaging facilities which are discussed here.

The basic primitive for making images is BitBlt designed by Dan Ingalls [18,37]. It operates on rectangular sub-regions of two *bitmaps* or two-dimensional arrays of pixels, setting each pixel of the target rectangle to some function of its old value and the value of the corresponding source pixel. Useful functions are

- constant black or white, which sets the target black or white;
- source, which copies the source to the target;
- merge, which adds black ink from the source to the target;
- xor, which reverses the target color where the source is black.

There is also provision for a *texture* source which is an infinite repetition of a 4 x 4 array of pixels; this provides various shades of gray, striping, etc. The Alto has a micro-coded implementation of BitBlt which is quite fast.

BitBlt is used for rearranging windows and for scrolling images in a window. In addition, a single BitBlt can draw an arbitrary rectangle, and in particular a horizontal or vertical line, or it can copy or merge the image of a character from a source image called a *font* which contains all the characters of a typeface. Arbitrary lines and curves are drawn by procedures that manipulate the target image directly, since the rectangles involved are too small for BitBlt to be useful. In many cases, however, the resulting images are saved away as templates, and BitBlt copies them to the screen image.

In the Alto system applications make images by using BitBlt directly, or by using procedures to paint a character string in a given font or to draw a line or spline curve. Curves, once computed, are stored in a chain encoding that uses four bits to specify the direction of the curve at each pixel. There is also a package that handles arrays of intensity samples, built by Bob Sproull, Patrick Baudelaire and Jay Israel; it is used to process scanned images. Cedar has a complete graphics package that handles transformations, clipping, and halftoned representations of sampled images within a uniform framework [59].

6 Applications

Most people who look at the Alto system see the applications which allow users to do many of the major tasks that people do with paper: preparing and reading documents with text and pictures, filing information, and handling electronic mail. This section describes the major Alto system applications that are not programming systems: editors, illustrators, and filing and mail systems.

6.1 Editors

The Bravo editor was probably the most widely used Alto application [25]. It was designed by Butler Lampson and Charles Simonyi, and implemented by Simonyi, Tom Malloy, and a number of others. Bravo's salient features are rapid screen updating, editing speed independent of the size of the document, what-you-see-is-what-you-get formatting (with italics, Greek letters and justified text displayed on the screen), and semi-automatic error recovery. Figure 2 shows a Bravo screen, illustrating the appearance of formatted documents. Later versions also have style sheets, which introduce a level of indirection between the document and its layout, so that the layout can be changed systematically by editing the style sheet; the document specifies "emphasis" and the style sheet maps it to "italic" or to "underlined" as desired.

Bravo is a rather large program, since it includes a software-implemented virtual memory for text and fonts; layout of lines and paragraphs in a variety of fonts with adjustable margins, tab stops, and leading; mapping from a point in the displayed text back to the document; incremental screen update; a screen display identical to the final printed copy; hard-copy generation for Ears and Press printers as well as daisy-wheel devices; and page layout. Later versions have a document directory, style sheets, screen display of laid-out pages, hyphenation, a remote terminal service, an abbreviation dictionary, form letter generation, and assorted other features.

Good performance in a small machine comes from the representation of edits to the text and of formatting information, and from the method for updating the screen. The text is stored as a table of *pieces*, each of which is a descriptor for a substring of an immutable string stored in a file. Initially the entire document is a single piece, pointing to the entire file from which it was read. After a word is replaced, there are three pieces: one for the text before the word, one for the new characters, which are written on a scratch file as they are typed, and one for the text after the word. Since binary search is used to access the piece array, the speed is logarithmic in the number of edits. This scheme was independently invented by Jay Moore [35].

Formatting information is represented as a property record of 32 bits for each character (font, bold, italic, offset, etc.); since consecutive characters usually have the same properties, they are stored in run-coded form in a table much like the piece table. Changes in formatting are represented by a sequence of *formatting operators* attached to each piece. Thus to find the font for a character it is necessary first to find its entry in the run-coded table, and then to apply all the formatting operators in its piece. But it takes only a few instructions per piece to make a 60,000 character document italic. When an editing session is complete, the document is written to a file in a clean form, so that the piece table doesn't keep growing.

To make screen updating fast, the screen image is maintained as a table of lines, each containing its bitmap and pointers to the characters used to compute it. This table is treated as a cache: when an edit is done, any entries which depend on characters changed by the edit are invalidated. Then the entire screen is recomputed, but the cache is first checked for each line, so that recomputation is done only if the line is actually different.

Bravo has a clumsy user interface. The Gypsy editor, built by Larry Tesler and Tim Mott, uses much of the Bravo implementation but refines the user interface greatly. Gypsy introduced a modeless user interface, editable screen display of bold and italics, and an integrated file directory that is modified by the same commands used for editing. All these ideas were later used in many parts of the Alto system. The parallel development of Bravo and Gypsy is an illustration of the many elements required in a good application. Both the refined implementation methods of Bravo and the refined user interface of Gypsy are important; each required several years of work by talented people.

The Star editor evolved from Bravo and Gypsy, with further refinement of the user interface [43] and the integration of graphics (see the next section). The Star interface design in turn had considerable influence on later versions of Bravo, from which Microsoft Word was then derived. It also influenced the Cedar document editor, Tioga, which is distinguished by its support of tree-structured documents (derived from Engelbart's system [13]), and by a fully programmable style-sheet facility which can do arbitrary computations to determine the format of a piece of text.

There are also simple text editors embedded in the Smalltalk and Mesa programming environments. These allow plain text files to be created, read, and modified. They are used for editing programs and for viewing *typescripts*, the output of programs that produce a sequence of unformatted lines.

6.2 Illustrators

The Alto system has a number of different illustrators for different kinds of pictures: pure bit-maps (Markup), spline curves (Draw), logic diagrams and other images involving only straight lines (Sil), and font characters (Fred). They all handle multi-font text as well, and all produce Press output for printing; most can show a screen image that exactly matches the printed image. Usually they work on only one picture at a time. None is integrated with a text editor; however, there is a separate batch program called PressEdit (written by William Newman), that can combine images from several Press files into a single file. PressEdit is used to assemble complete documents with text and figures; it has the drawback that the assembled document cannot be edited interactively. Several designs for integrated text and graphics editors stumbled over the limited size of the Alto.

William Newman's Markup was the first illustrator for the Alto [25]. It edits multi-page Press files, one page at a time, but it can only handle text and bitmap images, not lines or curves. Markup provides an assortment of brushes for painting into the bitmap, as well as operations for erasing and for moving or copying a rectangular region of the image. There are separate facilities for text strings, so that these can be printed at full resolution, since screen-resolution text looks terrible in hard-copy. Figure 4 is an example of the kind of picture that can be made. The Apple Macintosh's MacPaint is quite similar to Markup.

There are two very different illustrators that deal with synthetic graphics, in which the picture is derived from a set of mathematically defined curves. Draw, written by Patrick Baudelaire, builds up the picture from arbitrary lines and spline curves [25]. It has facilities for copying parts of the image and for applying arbitrary linear transformations. Curves can be of various widths, dashed, and fitted with various arrowheads. Thus very pretty results can be obtained; figure 5 shows some examples. The great variety of possible curves makes it difficult to use Draw for simple pictures, however, and it runs out of space fairly quickly, since it needs a full-screen bitmap (60 KBytes) and code for real arithmetic and spline curves, as well as the usual font-handling code, space for the fonts, the user interface etc. All of this must fit in the 128 KBytes of the Alto. Draw does not use the software virtual memory or overlaying techniques that allow Bravo to fit; these would have made the program quite a bit more complicated.

Sil was built by Chuck Thacker for making logic drawings [57]. Thus the main requirements were capacity for a complex drawing, a user interface well suited to experts who spend tens or hundreds of hours with the program, and a library facility for handling the images of logic components. Sil allows only characters and horizontal and vertical lines, which are easily handled by all the raster printers. There is a special font of bitmaps that represent and-gates, or-gates, resistors, and a few other specialized shapes, and a macro facility allows drawings of complex components to be built up from these. Since Sil drawings can be rather complex (see figure 6 for an example), redrawing the picture must be avoided. Instead, Sil draws a picture element with white ink to erase it, and in background redraws every element that intersects the bounding box of the erased element. This simple heuristic works very well. Somewhat to everyone's surprise, Sil is the illustrator of choice when curves are not absolutely required, because of its speed, capacity, and convenience.

For designing font characters with spline outlines Patrick Baudelaire built Fred. An Alto interfaced to

a television camera captures an image of the character, and the user then fits splines around it. The resulting shapes can be scan-converted at various resolutions to produce different sizes and orientations of the character for printing.

Sil and Fred are illustrators associated with batch-processing programs: a design-rule checker and wire-list generator for logic drawings made in Sil, and a battery of scan-conversion, font-editing and management software for Fred.

6.3 Filing

From its earliest days the Alto system has had the usual operating system facilities for managing stored information: directories can be listed and files renamed or deleted. This is quite clumsy, however, and many attempts have been made to provide a better interface. Most of these involve listing the contents of a directory as a text document. Lines of the listing can be selected and edited: deletion means deleting the file, changing the name renames the file, and so forth. Gypsy was the first system to use this scheme; it was followed by the Descriptive Directory System (DDS) built by Peter Deutsch [25], by Neptune built by Keith Knox, and later by Star's file folders. DDS and Neptune allow the user to control what is displayed by writing filters such as "memo *and not* report" which are applied to the file names. Star has a standard hierarchical directory system, which it manifests by allowing folders to appear in other folders.

Some other experiments with filing were more interesting. The Smalltalk browser is described in section 5.3 and illustrated in figure 1; it has been very successful. Cedar has a facility called Whiteboards, illustrated in figure 3, which deals with a collection of pages, each containing text and simple figures. A picture on one page can be a reference to another page; when the reference picture is clicked with the mouse, the page it points to is displayed. This is a convenient way to organize a modest amount of information. Finally, the electronic mail systems are heavily used for filing.

6.4 Electronic mail

The Alto system certainly did not introduce electronic mail, and in its early days the mail service was provided by Tenex. One of the most successful applications, however, was the Laurel user mail system built by Doug Brotz, Roy Levin, Mike Schroeder, and Ben Wegbreit [5]. Laurel provides facilities for reading, filing, and composing electronic messages; actual transport and delivery is handled by Grapevine or some other transport mechanism.

Laurel uses a fixed arrangement of three windows in a single column; see figure 7. The top window is a message directory, with one line for each message in the current folder. The other two windows hold the text of a message being read, and the text of a message being composed; the latter can be edited, and text can be copied from the middle window to the bottom. Above each window is a line of menu buttons relevant to that window; some of them have associated blanks which can be filled in with text when that button is clicked. A set of messages can be selected in the directory by clicking, and deleted, moved, or copied as a unit. Deleted messages are marked by being struck out in the directory; the actual deletion is only done when exiting Laurel, and can be undone until then.

Laurel can handle any number of message folders, and allows a message to be moved or copied from one folder to another. It also provides commands to forward or answer a message; these set up the composition window in a plausible way, but it is easy for the user to edit it appropriately. Message bodies are copied to the user's Alto from a mail server when the message is first retrieved. A message folder is represented as two Alto files, a text file for the messages and a table-of-contents file; there is a scavenger that reconstructs the latter from the former in case of trouble.

This apparently simple user interface is the result of about a man-year of design, and shows its value in the fact that most users never refer to the manual. This friendliness, together with the transparent view of the state provided by the message directory, and the high reliability of message retrieval and storage, made Laurel very popular. A variant of Laurel called Cholla is used as the backbone of a control system for an integrated circuit fabrication line; special messages called recipes contain instructions for the operator and the computer-controlled equipment, and messages are sent to the next station and to administrators and logging files as a job progresses through the system. The beauty of this system is that everything is immediately accessible to a person trying to determine the state or track down an error, and that the high reliability of the Grapevine transport mechanism makes lost data extremely unlikely.

Cedar has a mail facility called Walnut, patterned on Laurel, but using the standard Viewers windows and menus; it was built by Willie-Sue Haugeland and Jim Donahue. Walnut uses the Cedar entity-relationship database built by Rick Cattell to store messages; this turned out to be relatively unsuccessful, much to our surprise. It seems that a database

system does not have much to contribute to a mail system.

7 Conclusion

During the 1970's the Computer Science Laboratory and Learning Research Group at Xerox PARC built a complete personal distributed computing system: computers, display, local network, operating systems, programming environments, communication software, printing, servers, windows, user interfaces, raster graphics, editors, illustrators, mail systems, and a host of other elements. This Alto system did not have a detailed plan, but it was built in pursuit of a clear goal: to make computers better tools for people to think and communicate.

The Alto system has spawned a great variety of offspring. The Dorado system and the Xerox Star are the most direct line. Table 3 lists a number of commercial systems derived from the Alto. There have also been many in universities and research laboratories.

A dozen years of work have made it clear that these systems only begin to exploit the possibilities of personal distributed computing. Continuing progress in devices ensures that processors and networks will get faster, storage devices larger, displays clearer and more colorful, and prices lower. Compact discs, for example, allow half a gigabyte to be reproduced for a few dollars and to be on-line for a few hundred dollars; the impact will surely be profound.

These improvements and, even more important, a better understanding of the problems will lead to systems that are easier to program and use, provide access to far more information, present that information much more clearly, and compute effectively with it.

Acknowledgements

I am grateful to Bob Sproull, Peter Deutsch, David Boggs, Carol Peters and Cynthia Hibbard for comments on earlier versions of this paper, and to Ed McCreight and Adele Goldberg for supplying some of the figures.

CSL

Roger Bates	Dorado
Patrick Baudelaire	Draw, Fred, AIS, Puffin
Andrew Birrell	Grapevine
David Boggs	Alto OS, IFS, Scavenger, Dicentra
Jim Curry	BCPL
Peter Deutsch	XGP, Alto Lisp, DDS
Jerry Elkind	CSL manager
John Ellenby	Alto 2, Dover, Puffin
Chuck Geschke	Mesa
Jim Horning	Laurel, Cedar
Butler Lampson	Alto, Alto OS, Bravo, Mesa, Juniper, Dorado, Dandelion, Orbit, Interpress, Cedar
Roy Levin	Laurel, Grapevine, Wildflower, Cedar
Tom Malloy	Bravo
Ed McCreight	Alto, Exec, Dorado, design automation
Gene McDaniel	Alto OS, Dorado
Bob Metcalfe	Ethernet, Pup
Jim Mitchell	Mesa, Juniper
Jim Morris	Scavenger, Mesa, Juniper, Swat, Cedar
Severo Ornstein	Orbit, Dorado, Etherphone
Bill Paxton	Tioga
Paul Rovner	Cedar
Ed Satterthwaite	Mesa
Mike Schroeder	Laurel, Grapevine, Cedar
Charles Simonyi	Bravo, Juniper
Bob Sproull	Press, Alto OS, Chat, Orbit, Spruce, Interpress
Howard Sturgis	Juniper
Ed Taft	IFS, Pup, Dorado, Alpine
Bob Taylor	CSL Manager
Warren Teitelman	Lisp, Cedar
Chuck Thacker	Alto, mouse, Sil, design automation, Dorado, Dolphin
Ben Wegbreit	Laurel

SSL

Roger Bates	Mouse, Trident disk controller
David Boggs	Ethernet, IFS, Pup, FTP
Tim Mott	Gypsy, Officetalk
William Newman	Press, Markup, Officetalk
Bill Paxton	Officetalk, Tioga
Ron Rider	Ears
Larry Tesler	Gypsy, Smalltalk

LRG

Alan Kay	Smalltalk
Dan Ingalls	Smalltalk, Bitblt

Table 1: People and their projects

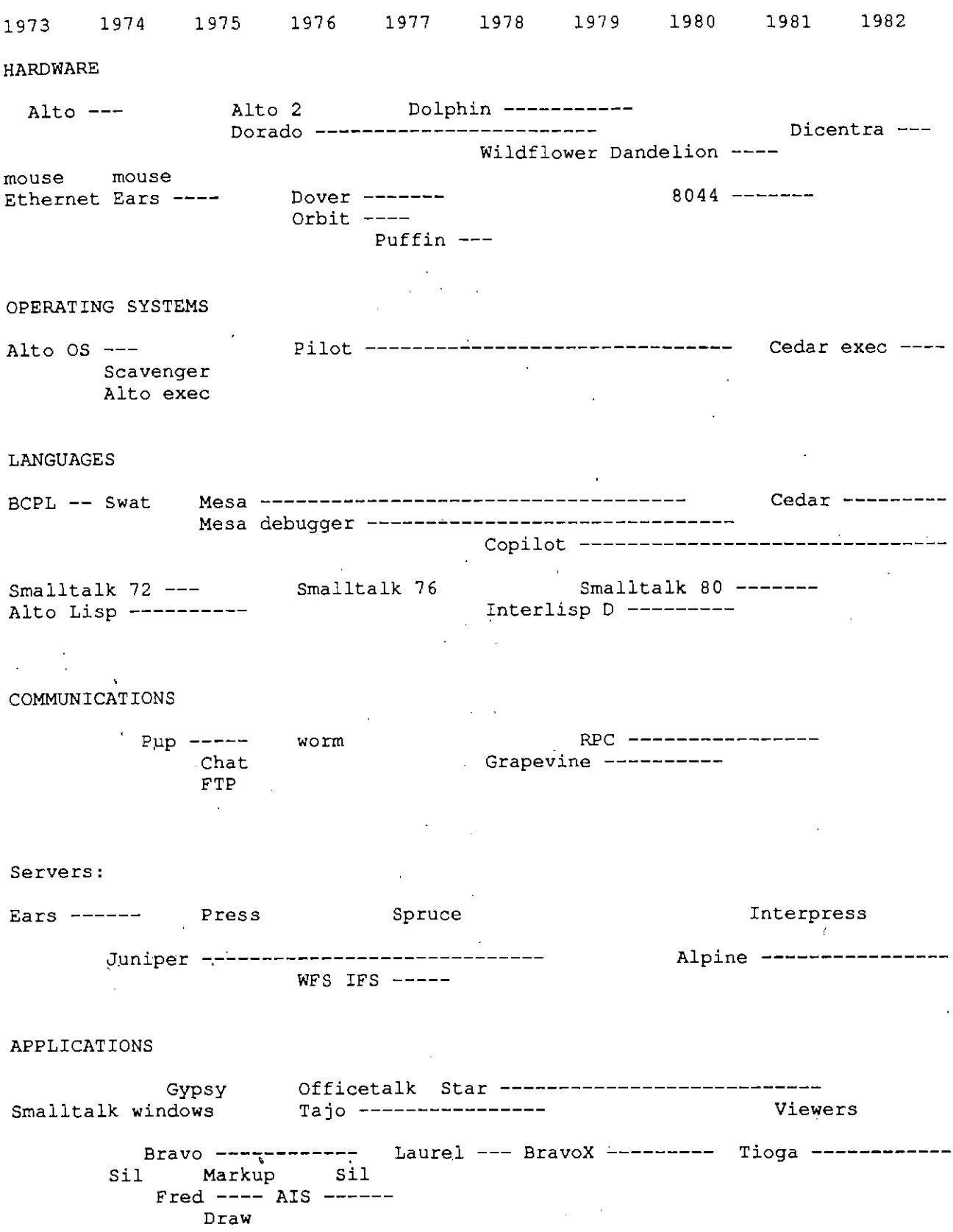


Table 2: Systems and chronology

Engineering workstations	Perq; Apollo; Sun; Tektronix; DEC; etc.
AI workstations:	MIT Lisp machine; Xerox 1100/1108/1132; Symbolic; LMI: etc.
Personal computers:	Apple Lisa, Macintosh; Xerox 8065.
Office workstations	Convergent NGen; Xerox 8010; Apple Macintosh; Grid; etc.
Graphics terminals	BBN Bitgraph; Bell Labs Blit; etc.
Local area networks	Ethernet/IEEE 802.3.
Networks protocols	ARPA IP/TCP; Xerox Network Services, Clearinghouse; etc.
Laser Printers	Xerox 9700, 5700, 8044; Imagen; Apple Laserwriter; etc.
Printing interfaces	Xerox Interpress; Adobe Postscript; etc.
Servers	3Com file server, Xerox 8044 and Apple Laserwriter print servers, etc.
User interfaces	Xerox 8010; Apple Macintosh; Microsoft Windows; etc.
Editors	Apple Macwrite; Microsoft Windows; etc.
Illustrations	Xerox 8010; Apple MacPaint, MacDraw; etc.

Table 3: Commercial systems descended from the Alto

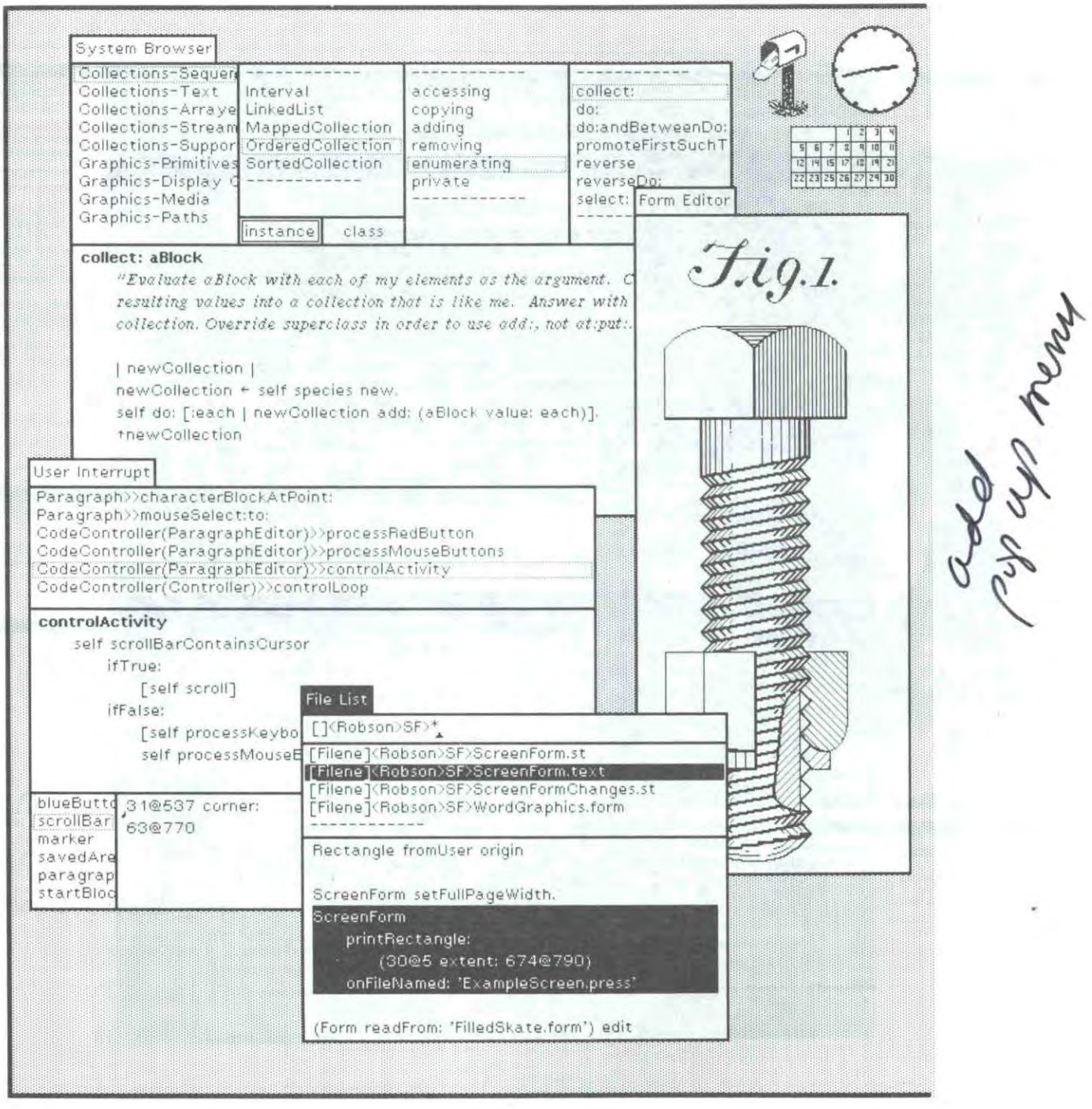


Figure 1: A Smalltalk screen; note the overlapping windows and the browser

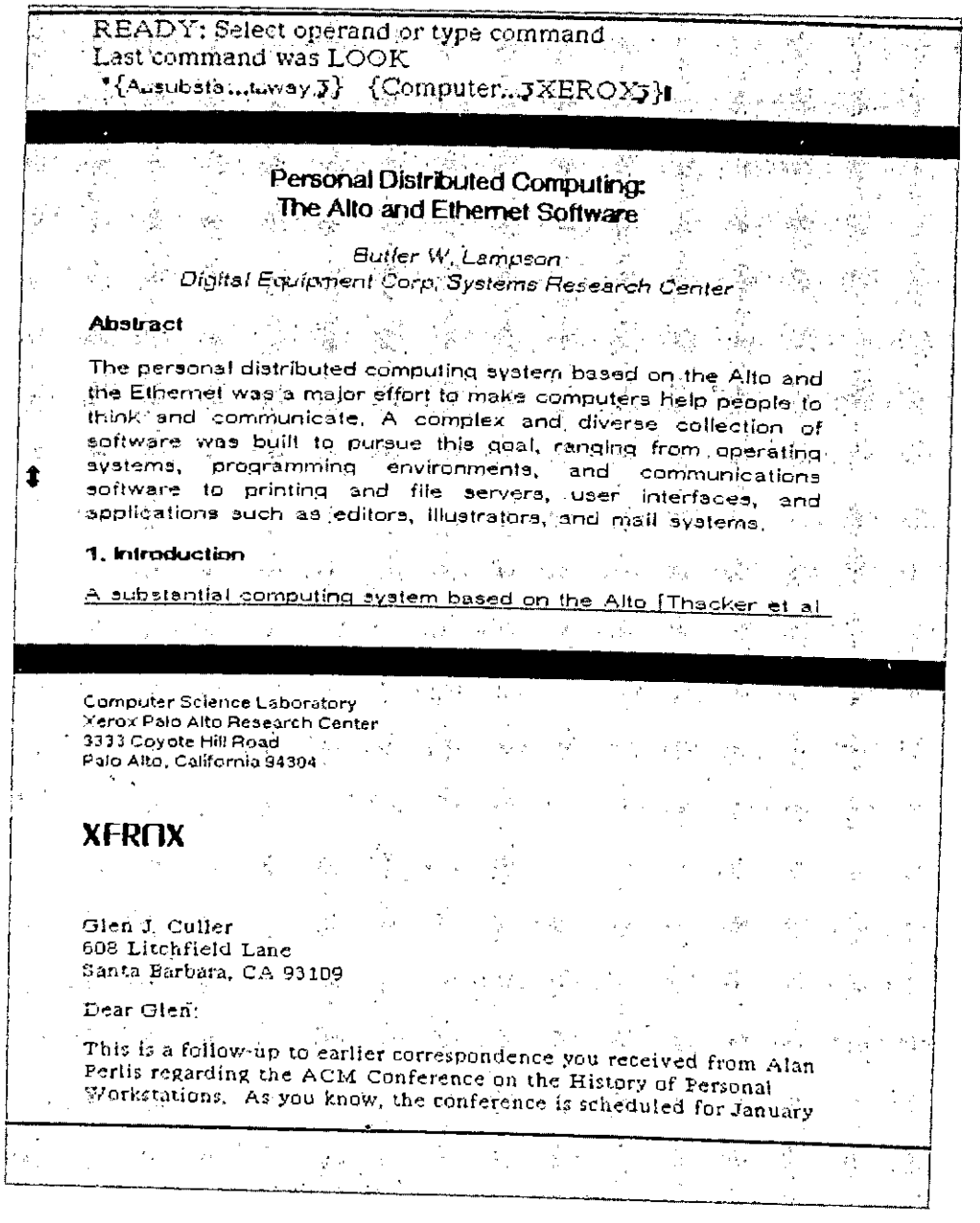


Figure 2: A Bravo screen; note the formatted document

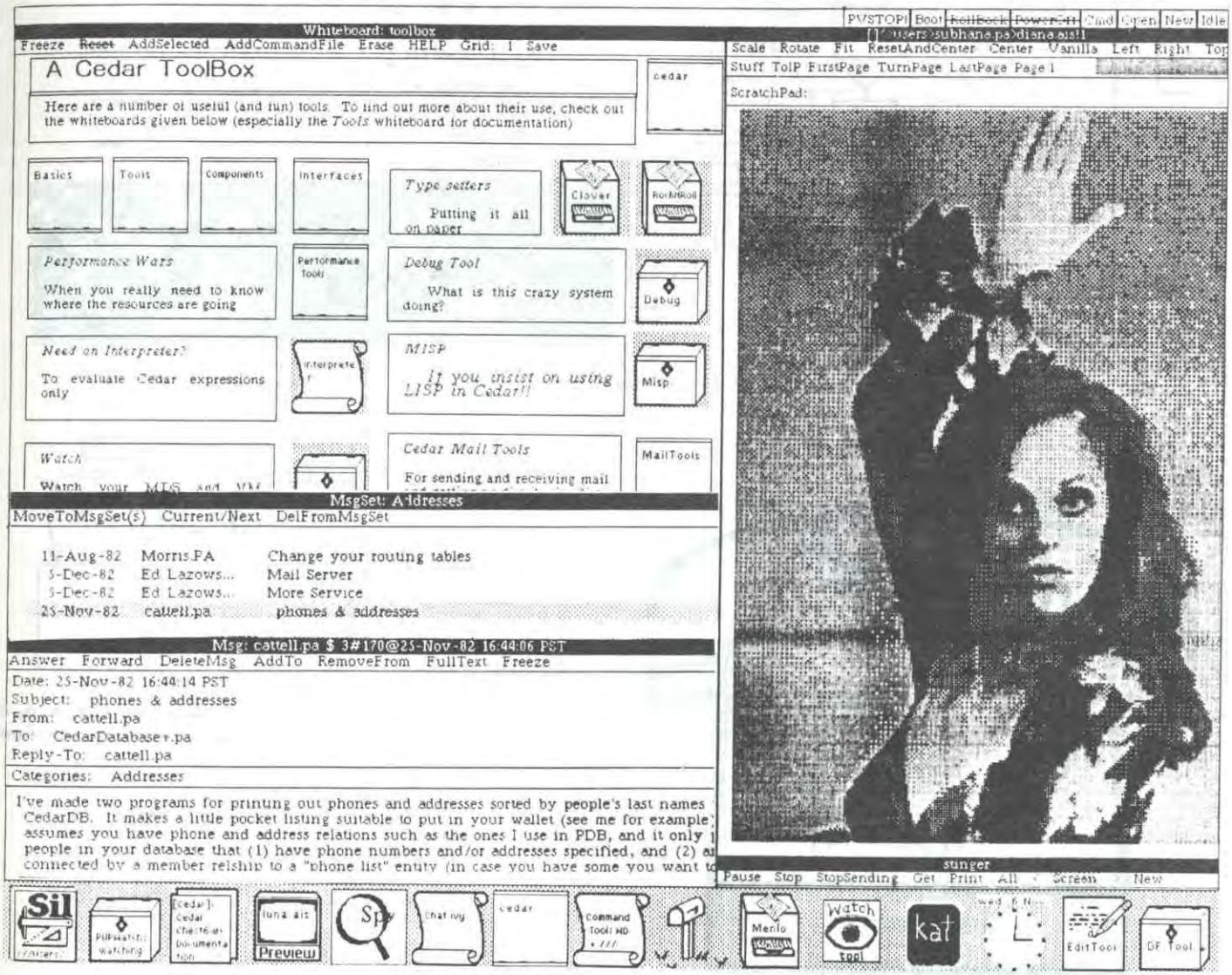


Figure 3: A Cedar screen; note the two-column tiling, the icons and the whiteboard

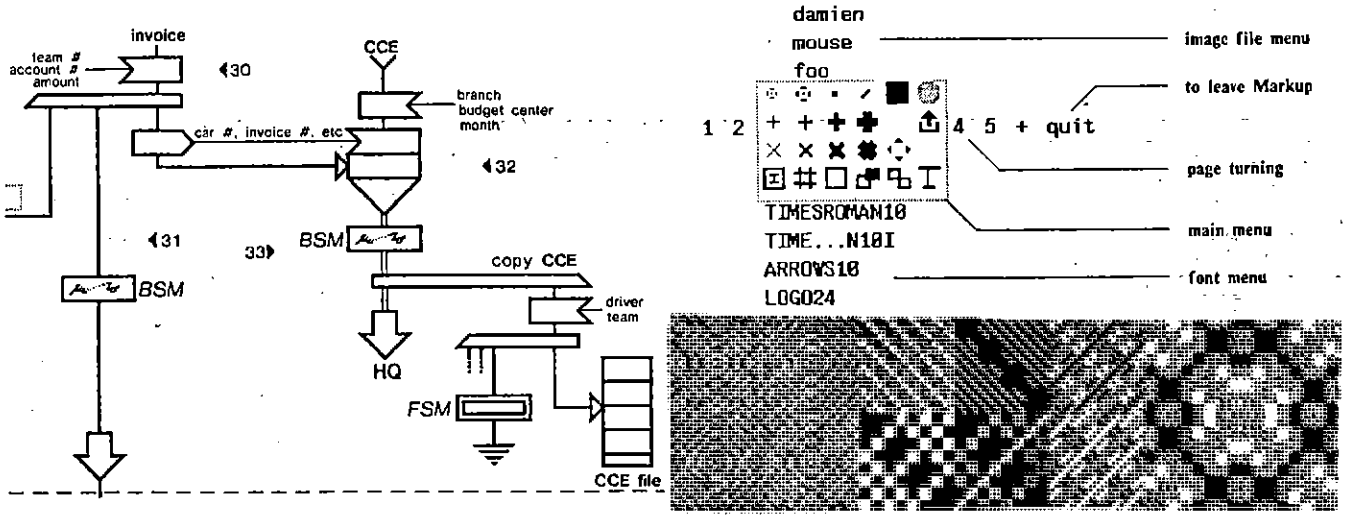


Figure 4: Typical Markup pictures

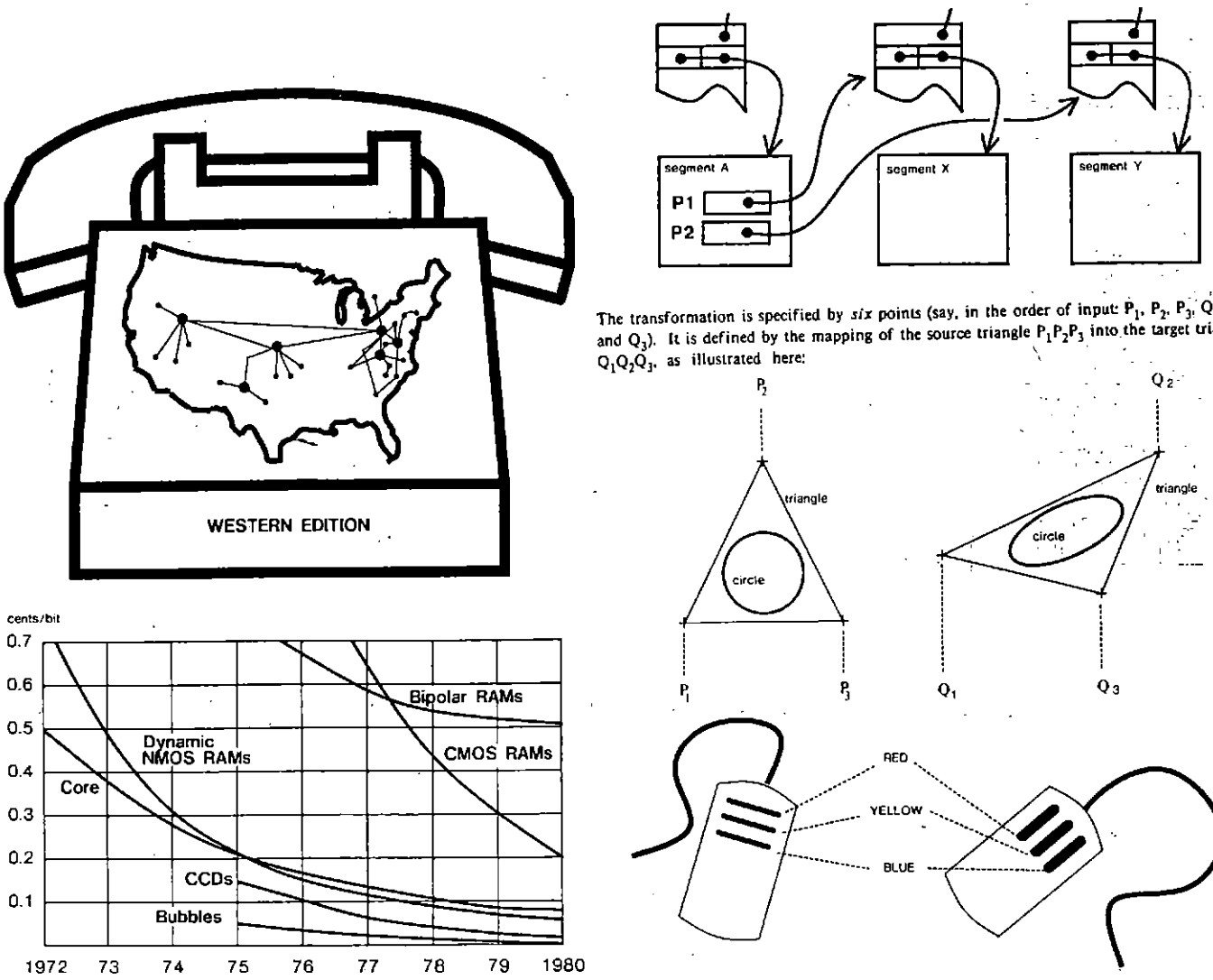


Figure 5: Typical Draw pictures

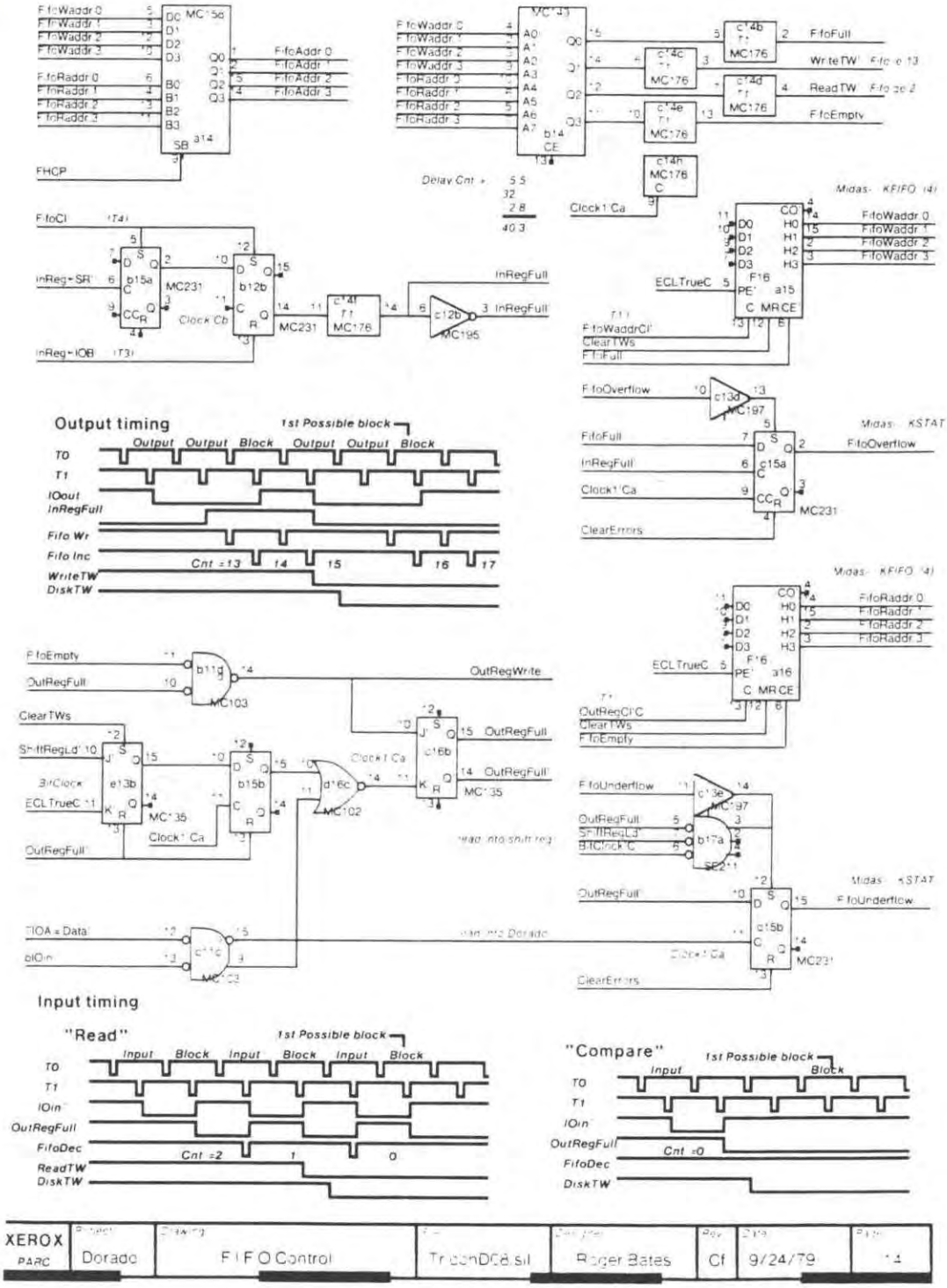


Figure 6: A logic drawing done in Sil

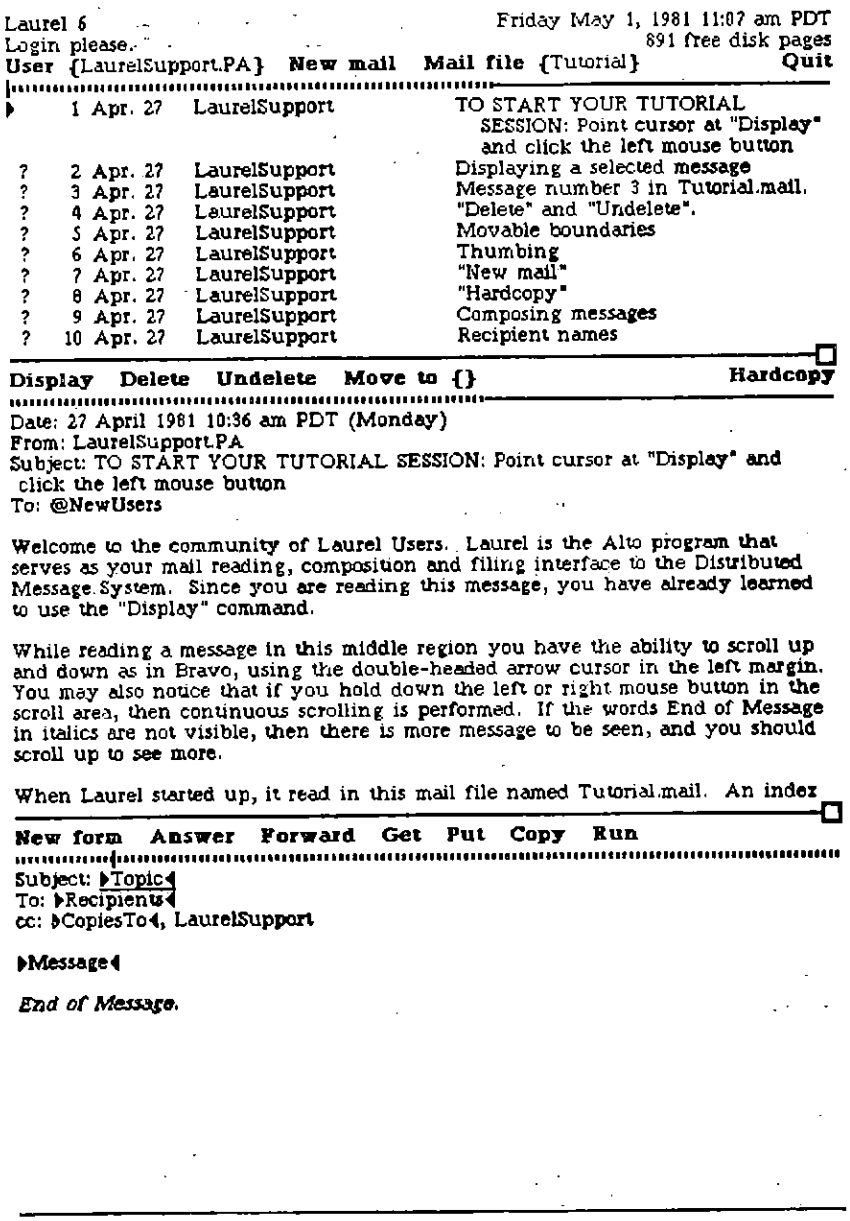


Figure 7: A Laurel screen

References

[1] A.D. Birrell et al. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[2] A.D. Birrell and B.J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[3] D.G. Bobrow et al. Tenex: A paged time-sharing system for the PDP-10. *Communications of the ACM*, 15(3):135–143, March 1972.

[4] D.R. Boggs et al. Pup: An internetwork architecture. *IEEE Transactions on Communications*, 28(4):612–624, April 1980.

[5] D.K. Brotz. *Laurel Manual.* Technical Report CSL-81-6, Xerox Palo Alto Research Center, 1981.

[6] M.R. Brown et al. The Alpine file system. *ACM Transactions on Computer Systems*, 3(2), November 1985.

[7] R.R. Burton et al. Interlisp-D: Overview and status. In *Proc. Lisp Conference*, Stanford, 1980.

[8] P.A. Crisman, editor. *The Compatible Time-Sharing System: A Programmer's Guide.* MIT Press, 2nd edition, 1965.

[9] L.P. Deutsch. Experience with a microprogrammed Interlisp system. *IEEE Transactions on Computers*, C-28(10), October 1979.

[10] L.P. Deutsch. A Lisp machine with very compact programs. In *Proc. 3rd International Joint Conference on Artificial Intelligence*, Stanford, 1973.

[11] L.P. Deutsch and E.A. Taft. *Requirements for an experimental programming environment.* Technical Report CSL-80-10, Xerox Palo Alto Research Center, June 1980.

[12] D.C. Engelbart. The augmented knowledge workshop. In *Proc. ACM Conference on the History of Personal Workstations*, January 1986.

[13] D.C. Engelbart and W.K English. A research center for augmenting human intellect. In *Proc. AFIPS Conference*, pages 395–410, 1968.

[14] E.R. Fiala. The MAXC systems. *IEEE Computer*, 11(5):57–67, May 1978.

[15] C.M. Geschke et al. Early experience with Mesa. *Communications of the ACM*, 20(8):540–553, August 1977.

[16] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[17] J. Gray et al. The recovery manager of the System R database manager. *ACM Computing Surveys*, 13(2):223–242, June 1981.

[18] D. Ingalls. The Smalltalk graphics kernel. *Byte*, 6(8):168–194, August 1981.

[19] D.H. Ingalls. The Smalltalk-76 programming system: Design and implementation. In *Proc. 5th ACM Symposium on Principles of Programming Languages*, pages 9–16, January 1978.

[20] R.K. Johnsson and J.D. Wick. An overview of the Mesa processor architecture. *ACM Sigplan Notices*, 17(4):20–29, April 1982.

[21] A.C. Kay. *The Reactive Engine.* PhD thesis, University of Utah, 1969.

[22] A.C. Kay and A. Goldberg. Personal dynamic media. *IEEE Computer*, 10(3), March 1977.

[23] B.W. Kernighan and R. Pike. *The Unix Programming Environment.* Prentice-Hall, 1983.

[24] D.E. Knuth. *TeX and Metafont: New Directions in Typesetting.* Digital Press and American Mathematical Society, 1979.

[25] B.W. Lampson, editor. *Alto User's Handbook.* Xerox Palo Alto Research Center, 1976.

[26] B.W. Lampson et al. A user machine in a time-sharing system. *Proc. IEEE*, 54(12):1744–1766, December 1966.

[27] B.W. Lampson and K.A. Pier. A processor for a high-performance personal computer. In *Proc. 7th Symposium on Computer Architecture*, pages 146–160, ACM Sigarch/IEEE, May 1980.

[28] B.W. Lampson and D.D Redell. Experience with processes and monitors in Mesa. *Communications of the ACM*, 23(2):105–117, February 1980.

[29] B.W. Lampson and R.F. Sproull. An open operating system for a single-user machine. *ACM Operating Systems Review*, 13(5), November 1979.

[30] H.C. Lauer and E.H. Satterthwaite. The impact of Mesa on system design. In *Proc. 4th International Conference on Software Engineering*, pages 174–182, September 1979.

[31] P.J. Leach et al. The architecture of an integrated local network. *IEEE Journal on Selected Areas of Communication*, SAC-1(5):842–856, November 1983.

[32] J. Licklider. Man-computer symbiosis. *IRE Trans. Human Factors in Electronics*, HFE-1:4–11, March 1960.

[33] J.M. McQuillan and D.C. Walden. The Arpanet design decisions. *Computer Networks*, 1(5):243–289, September 1977.

[34] J.G. Mitchell and J. Dion. A comparison of two network-based file servers. *Communications of the ACM*, 25(4):233–245, April 1982.

[35] J.S. Moore. *The TXDT Package - Interlisp Text Editing Primitives.* Technical Report CSL-81-2, Xerox Palo Alto Research Center, January 1981.

[36] R.M. Needham and A.J. Herbert. *The Cambridge Distributed Computing System.* Addison-Wesley, 1982.

[37] W.M. Newman and R.F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, 2nd edition, 1979.

[38] D.D. Redell et al. Pilot: An operating system for a personal computer. *Communications of the ACM*, 23(2):81–92, February 1980.

[39] H. Rheingold. *Tools for Thought.* Simon and Schuster, 1985.

[40] M. Richards. BCPL: A tool for compiler writing and system programming. In *Proc. AFIPS Conference*, pages 557–566, 1969.

[41] M.D. Schroeder et al. A caching file system for a programmer's workstation. *ACM Operating Systems Review*, 19(5), December 1985.

[42] J.F. Shoch and J.A. Hupp. Notes on the 'worm' programs–some early experiences with a distributed computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[43] D.C. Smith et al. The Star user interface: An overview. In *Proc. AFIPS Conf.*, pages 515–528, 1982.

[44] R.F. Sproull. *Introduction to Interpress.* Xerox Printing Systems Division, 1984.

[45] R.F. Sproull. Raster graphics for interactive programming environments. *Computer Graphics*, 3(3), July 1979.

[46] R.M. Stallman. EMACS: the extensible, customizable self-documenting display editor. In *ACM Sigplan Notices*, pages 147–156, June 1981.

[47] J.E. Stoy and C. Strachey. OS6–an experimental operating system for a small computer. *Computer Journal*, 15(2 and 3), May and August 1972.

[48] I. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. AFIPS Conf.*, pages 329–346, 1963.

[49] R.E. Sweet. The Mesa programming environment. *Sigplan Notices*, 20(7):216–229, July 1985.

[50] D. Swinehart et al. WFS: A simple shared file system for a distributed environment. *ACM Operating Systems Review*, 13(5), November 1979.

[51] D.C. Swinehart et al. The structure ofCedar. *Sigplan Notices*, 20(7):230–244, July 1985.

[52] W. Teitelman. A display-oriented programmer's assistant. In *Proc. 5th International Joint Conference on Artificial Intelligence*, pages 905–917, 1977.

[53] W. Teitelman. A tour through Cedar. *IEEE Software*, 1(4), April 1984.

[54] W. Teitelman et al. *Interlisp Reference Manual.* Technical Report, Xerox Palo Alto Research Center, 1978.

[55] W. Teitelman and L. Masinter. The Interlisp programming environment. *IEEE Computer*, 14(4):25–34, April 1981.

[56] C.P. Thacker. Personal distributed computing: The Alto and Ethernet hardware. In *ACM Conference on the History of Personal Workstations*, January 1986.

[57] C.P. Thacker. SIL—a simple illustrator for CAD. In S. Chang, editor, *Fundamentals Handbook of Electrical and Computer Engineering, Volume 3*, pages 477–489, Wiley, 1983.

[58] C.P. Thacker et al. Alto: A personal computer. In Siewiorek et al., editors, *Computer Structures:*

Principles and Examples, chapter 33, McGraw-Hill, 1982. Also CSL-79-11, Xerox Palo Alto Research Center (1979).

[59] J. Warnock and D.K. Wyatt. A device independent graphics imaging model for use with raster devices. *Computer Graphics*, 6(3), July 1982.

[60] J.E. White and Y.K. Dalal. Higher-level protocols enhance Ethernet. *Electronic Design*, 30(8):31–41, April 1982.

Butler et al
need to consider
Gordon Bell's
questions

---

ask more designers
from Xerox
influenced

Rob Pike
Steve Tibbs
Nick Wirth

THE LINC WAS EARLY AND SMALL

Wesley A. Clark

Clark, Rockoff, and Associates
One Sherman Square, New York, NY 10023

SUMMARY

The LINC represents one of the earliest attempts to put the stored program computer into the form of a general instrument for laboratory use. In a deliberate departure from the technology of Timesharing then just beginning nearly two decades of development, the LINC was designed for use by individual experimenters and thus anticipated features of the modern personal computer and personal workstation. Built at M.I.T. in 1962, its immediate forebears were the TX-0, ARC-1, and L-1 computers, in turn direct descendents of the M.I.T. Whirlwind and MTC computers. Of course the LINC in its day was neither personal computer nor personal workstation but simply the LINC.

The LINC was an outgrowth of interactions between two M.I.T. groups of scientists and engineers: the Communications Biophysics Laboratory interested in the quantification of neuroelectric activity, and the Lincoln Laboratory Digital Computer Group engaged in the development of advanced computers. Twelve LINCs were placed initially in biomedical research laboratories across the country under a unique NIH/NASA-sponsored evaluation program. Ultimately more than 1200 LINC or LINC variants were manufactured commercially for worldwide use. The basic system design went on to influence the design of the DEC PDP-4 and PDP-5 computers, which in turn helped to pave the way to the PDP-8.

This paper is a revision and expansion of the paper "The Origins of the LINC", Proceedings of the IEEE Seventh Annual Conference of the Engineering in Medicine and Biology Society, ©IEEE 1985; additional figures and tables are taken from Computers in Biomedical Research Vol 2, R. W Stacy and B. D. Waxman (eds), ©Academic Press 1965. Permission of the publishers is gratefully acknowledged.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0133 $00.75



1963: The author and Mr. Charles E. Molnar with a LINC. The photograph was taken in Cambridge following the first phase of LINC Evaluation Program activities in which groups of scientists were given special training at a one-month summer course at M.I.T.



1981: The author and Dr. Charles E. Molnar with a LINC in a recreation of the above scene following a Digital Museum Lecture (see wall poster). Eighteen years grayed us both. Several of the original LINCs, also graying, were still in use in various parts of the world. (Photo courtesy of Digital Equipment Corporation)

INTRODUCTION

Twenty-two-plus years ago — in September, 1963 — the last of about twelve freshly assembled LINCs was safely delivered by moving van from M.I.T. to its new home in California. By itself, perhaps, this would not have been considered particularly noteworthy. But the event marked the successful completion of Phase I of a remarkable and unprecedented program. The twelve or so LINCs assembled during the very hot Cambridge summer of '63 had been put together by their new owners themselves. Each of these pioneers would take full responsibility for trial operation of the instrument as a workstation in his own biomedical research laboratory, and was expected to use it in the mode we have come to call, in Alan Kay's powerful paradigm, personal computing.

The participants in this innovative NIH/NASA-supported program had been selected on the basis of how suitable such a laboratory instrument would be in their ongoing or planned research activities. Each participant had agreed to assume individual responsibility for using and maintaining the machine throughout a subsequent eighteen-month period that defined Phase II of the program. The LINC Evaluation Program, as it was called, had been announced in *Science* and in descriptive brochures sent to academic and research institutions around the country. Candidates had been invited to submit proposals responsive to the scientific and technical objectives of the program. Computer programming or hardware experience was not required; the necessary fundamentals were to be learned in a one-month intensive training and kit assembly program to be conducted at M.I.T. by the LINC design team. Ultimate disposition of the assembled instruments was to be determined upon completion of Phase II.

More than seventy proposals had been received and studied by a specially created LINC Evaluation Board composed of distinguished scientists and engineers from research institutions across the country. After careful review the Board had finally accepted just twelve. These represented the fields of psychology, physiology, genetics, pharmacology, biophysics, and neurology. The selected participants had been divided into two groups to be convened in July and August respectively, with each principal investigator to be accompanied by one associate or colleague.

By mid-spring of 1963, all of the personal commitments had been made and much of the LINC hardware was in manufacture. But apart from one operating system/assembly program that had been developed and tested by simulation on a larger machine, the Lincoln Laboratory TX-2, the only LINC software in existence consisted of a few programs that had been written for an earlier demonstration prototype. With only two months to go, the final design of the LINC itself had not yet been completed.

How had all this come about? What was the LINC and what were its design objectives? Why did the field of biomedical research play such an important role in its development? I will try to recount something of the early history in answer to these questions. Like all such accounts of the introduction of a complex new instrument, the story is one of people, ideas, machines, and institutions and spans many years. As a principal instigator and designer of the LINC my perspective is limited to the events and circumstances that shaped only the earliest chapters. The full story must be left to others.

In what follows I will try to underscore and develop the title's assertion that the LINC, as personal workstations go, was early and small [1]. Early is evident; small, on the other hand, must be understood in the context of the memory capacity and the physical size of computers of the period. Since the machine's characteristics have been well documented (though not in the computer literature) I have indulged an urge to reminisce mostly about the people and events that really made the LINC experience such a rich and rewarding one. Some parts of the historical background may seem to be unnecessary embellishments, but I have included them in the hope that they will help in tracing the development of ideas and themes. The *dramatis personae* comprise extensive lists in both the computer and biomedical disciplines, and for the sake of simplicity I have kept professorial titles to a minimum and have arbitrarily 'Dr.'ed only those players who appear in character from the biomedical side of the stage. I hope that my friends will forgive me for this, as well as for any miscrediting of ideas and any errors or omissions of fact that may have settled into my own memory over the years.

BACKGROUND

Interacting with Whirlwind and MTC

Al Perlis and Doug Ross and I fondly remember the M.I.T. Digital Computer Laboratory Whirlwind computer, the machine on which I learned what little anyone knew about programming in 1952. One walked into it. What now sits comfortably on a small desktop, in those days required an entire room for the control consolery alone. Programming for its small electrostatic memory (1024 16-bit words on a good day) was a primitive affair carried out with the aid of heavily ruled coding forms on which to write out absolute-address instructions and octal numbers. [Confronted by the prospect of having to program in such a manner, Al immediately designed a new improved form that consisted of a large sheet of blank paper.] But its early users did indeed walk into the control room and for their assigned block of time, typically fifteen minutes or so depending on the time of day, the entire machine was theirs. In this regard the use of Whirlwind as a workstation of sorts was no different from that of many other early computers, especially in the universities. What makes the Whirlwind computer important in the history of the personal workstation is the fact that it was the first really high-speed machine with powerful interactive display capabilities [2].

But it was MTC that really drove the point home. MTC, the Memory Test Computer, was designed and built by Harlan Andersen and Ken Olsen to provide a working computer system in which to try out

the first ferrite core memory, a vacuum-tube driven 1K 16-bit unit engineered by Bill Papian then a graduate student under Jay Forrester [3]. Blocks of time measured in hours were available and the entire machine, quite similar in architecture to Whirlwind, now occupied only a single large room. Bristling with toggle-switches, pushbuttons, and indicator lights and provided with audio output as well as versatile CRT displays, it made interactive use a quite lively and memorable experience.

I suppose the idea of using even a very large computer such as the 1953 MTC just as one would use any laboratory tool was introduced to me by Belmont Farley, my teacher and collaborator in early computer simulation studies of neuron-like networks [4]. Belmont, a physicist by training and a man of considerable vision and strongly held convictions, joined the Digital Computer Laboratory shortly after my own move to M.I.T. to learn computerology. [I had almost become a physicist myself at one time, in what I now consider a close call; but that's another story.]

Belmont and I spent enormous numbers of hours interacting with MTC and with one another (he talked, I listened) as we tried out ideas, modified parameters, and studied displays of the simulated behavior of our strange little networks, grateful for such extensive access to such a powerful if not yet completely reliable machine. [A self-restarting test program was left running on MTC whenever the machine was otherwise unoccupied. In those pre-regulation days you could tune a radio to the frequencies being broadcast by MTC's long open-wire bus and hear the test program running, or trying to run, from several city blocks away. Sometimes when approaching by automobile for an extra-long simulation run in the dead of night the signal was clear enough for a decision to turn back home if things weren't going well.]

It was in these sessions that I began to learn from Belmont many of the basic attitudes toward computers that I hold firmly to this day: *Computers are tools; convenience of use is the most important single design factor — Big computers are for big jobs; small computers, for small jobs — Separate personal files are safer than files held in a common shared space — Digital computers should handle analog signals as well* — and so on.

In what would become a very important liaison, Belmont and I began to attend the seminars of the Communications Biophysics Laboratory, CBL, under Prof. Walter Rosenblith, later Provost of M.I.T. and presently Foreign Secretary of the National Academy of Sciences. Walter's group was concerned with the problem of quantifying neuroelectric phenomena. It was quite interdisciplinary in its approach and attracted students, post-doctoral fellows, and visiting scientists from many parts of the world. CBL had been developing various signal processing techniques. Much of CBL's work involved auto- and cross-correlation analyses based on data recorded on FM tape and then processed by analog devices. Work was just beginning on the analysis of the electrical signals from single neurons.

Our interaction with CBL started slowly. The Digital Computer Laboratory was absorbed into the recently established Lincoln Laboratory to become a component of Lincoln's massive program in air defense. Our base of operations shifted fifteen miles west to Lexington and we were assigned to a new group charged with advanced computer development together with Ken Olsen, Bill Papian, Dick Best, Jim Forgie, John Francovich, Ben Gurley, and many other members of the former Digital Computer Laboratory staff. Belmont and I somehow managed to keep up our pilot studies in network simulation with much travel back and forth to CBL and to the MTC in Cambridge.

Bill Papian and others, bolstered by the immediate success of the 1K MTC memory (shortly thereafter replaced by a 4K unit) now began to build the largest core-memory array thought feasible, a 64K 36-bit unit. This array would be thirty-six times the storage capacity of the 4K MTC array and would certainly constitute a big step in computer memory technology. In my new capacity as leader of a small sub-group in logic design I made sure that its potential for further simulation and modeling work was kept in mind.

The Lincoln TX-0 Computer

It was a time of transistors. Ken Olsen, in charge of the sub-group in advanced engineering development, joined me in proposing the construction of a vacuum-tube machine of considerable size, the TX-1, as a suitable vehicle for testing the large memory array [5]. When this idea was turned down by Laboratory management (wisely enough) Ken and others began to develop transistor circuits in earnest. I began the logic design of a new machine, the TX-2, which would have about the same architecture as that of the rejected machine but would take advantage of the new circuits. It is interesting to note that one of the main architectural features of the TX-1/TX-2 design — multiple program counters used in what I badly termed multi-sequence programming [6,7] — was later employed to great advantage by Chuck Thacker in the design of the Alto workstation [8].

I proposed that we first build a much smaller, primitively simple computer that I had designed and named the TX-0 [3,9]. [We had already used up "1"; later Ken and I would say that we didn't build the TX-1 because we didn't like its color scheme.] The 18-bit TX-0 would use only half of the large memory array and would be quite simple in logical structure, maximally RISCy, one might say today. When the second half of the array was completed the full array would then be installed in the TX-2 on which design and construction would have been proceeding. The TX-0 would then be retrofitted with a new memory unit of some kind and serve as a front end processor to the TX-2, or perhaps simply be dismantled. In the meantime it would have served in early evaluation of the new memory and transistor circuits and would have been more than adequate to the task of network simulation as well as to other tasks more directly relevant to the Lincoln effort. Not a bad plan for an R&D program. It was adopted. The TX-0 was built and a three-year effort to design and build the TX-2 was begun [10]. In addition to powerful CRT display units designed by Ben Gurley, the consoles of both machines would ultimately bristle in the best M.I.T. tradition.

The simulation work Belmont and I were carrying out had long had the interest and support of Bill Papian. Appointed Leader of the Advanced Development Group, Bill actively encouraged further strengthening of the interaction with the CBL group. Students and scientists from Rosenblith's laboratory began to try their hand at digital processing of neuroelectric data on the TX-0 and later on the TX-2. One of these students was Charles Molnar, who was to become a primary catalytic agent in promoting sound relationships within and between the two groups. Charlie, I soon learned, was extraordinarily competent and talented both as an electrical engineer and as a student of biophysics. He shared the conviction that computer tools would play an important part in what was beginning to be called "the biomedical sciences" and proceeded to master the operational and technical details of both the TX-0 and the unfinished TX-2. Charlie and I began to develop a basis of collaboration that would lead to the LINC and to many other developments over the years.

As the second half of the large memory array neared completion it became time to decide what to do with the TX-0. Bill Papian suggested replacing its memory with the group's new transistor-driven 4K memory unit (about the same capacity as that of the earlier vacuum-tube-driven MTC unit) and then moving the entire machine to the M.I.T. campus: This would give a wider group of students and faculty an opportunity to experience the kind of hands-on interactive computer use that characterized our work at Lincoln Laboratory in the tradition we were carrying forward from Whirlwind and MTC.

By that time both of the Cambridge machines, Whirlwind and MTC, had been completely committed to the air defense effort and were no longer available for general use. The only surviving computing system paradigm seen by M.I.T. students and faculty was that of a very large International Business Machine in a tightly sealed Computation Center: the computer not as *tool*, but as *demigod*. Although we were not happy about giving up the TX-0, it was clear that making this small part of Lincoln's advanced technology available to a larger M.I.T. community would be an important corrective step. The machine was then being used in some very exciting work in interactive programming by Jack Gilmore (unfortunately never published) that anticipated much of what we do today, but Bill's argument that the group's attention would be shifting to the TX-2 prevailed. Arrangements were made to replace the memory, enrich the instruction repertoire, and move the modified TX-0 to the campus in Cambridge.

But where exactly should it go? The logical place, it seemed to me, was Walter Rosenblith's laboratory. After all, CBL was then using the TX-0 computer more actively than any other group of visitors from the campus and expanded use by George Gerstein, Charlie, and other members of CBL was expected. To my surprise Walter declined to accept the responsibility. I therewith gained an important insight about what would work in a small laboratory setting and what would not. It seems in retrospect that the TX-0, small as it was for its time and demonstrably useful in CBL research, was still too much "the computer" and not sufficiently "an instrument". Its care and feeding and the constant need for justification might well have compromised CBL's research objectives. Walter's technical judgement, as usual, was correct.

The modified TX-0 eventually found a home under the larger umbrella of the Electrical Engineering Department, where it was used heavily by CBL and other groups and helped to train several generations of students in the interactive use of computers on-line. Somewhat later it was joined and largely upstaged by a PDP-1 computer (in some sense a production version of the modified TX-0) that had been donated to M.I.T. by the Digital Equipment Corporation and would later be used in some of the earliest experiments in Timesharing [3].

Timesharing had by then captured the imagination of much of the M.I.T. computer science community (not yet a science, of course; it would not be so until a few years later, when Al Newell, Al Perlis, and Herb Simon said, *Let there be Computer Science*, and there was). This seemed to me a bad idea and still does, the captive imagination aspect no less than the underlying premise itself. Improved access, yes; thrashing competition and waste, no. This is not to deny that Timesharing resulted in a huge and productive impetus to computer science at a critical time. Certainly it has given much gainful employment to computer scientists and manufacturers and salesmen over the past twenty years or so. But it seems to me that we could have done better than to divert so much of our attention and resources to trying to make good the promise of a patently unattainable "sensible simultaneity" for all. [I may not always be right, but at least I am consistent. When working out the multi-sequence programming idea some years earlier I had considered and rejected the notion of competitive time-sharing by independent human users as grossly inefficient; much later at the dawn of the Timesharing Era I had no qualms about turning down a suggestion of Ed Fredkin's to set up such a system for the multi-sequence TX-2 (much to his disappointment and, as I recall, disgust); in meetings of the M.I.T. Long Range Study Committee in 1961 I objected (to no effect) that campus-wide Timesharing so enthusiastically being rationalized as a panacea would not be able to deal with real-time work such as CBL's and moreover would inhibit the development of any interactive computing that involved complex displays. The M.I.T. Study Committee Chairman and I both declined to sign an otherwise unanimous final report. He thought it didn't promise nearly enough. Walter urged me to write a minority report, but I knew a steam roller when I saw one and went on to other matters.]

The ARC and the L-1

Not long after the decision to move TX-0 was made, Charlie and I met with Moise Goldstein and Robert Brown of CBL. The subject of discussion was the possibility of building a digital device to extract stimulus-evoked neuroelectric potentials from a background of unrelated electrical activity by means of a summation technique then being tried at CBL on analog equipment. Aha! A digital instrument for Walter! I immediately agreed to take on the design task.

The logic design of the instrument fell into place readily enough. I simply used many of the transistor logic plug-in modules already designed by Ken Olsen and Dick Best for the TX-2, and my colleague Henry Zieman neatly solved the problem of memory by working out the details of a very small 256-word core array. An 18-bit structure seemed about right for the job and nicely matched that of the TX-0 with which the instrument might later be in communication. I designed two simple operating modes to provide response averaging and amplitude histogram compilation, both modes to be wired into the control circuits. With electronics, control switches, lights, CRT, plotter, and paper-tape punch, the whole thing — even without an analog-to-digital converter — required a cabinet about the size of two refrigerators turned on their sides. Today we would think it a machine of rare ugliness (as an unkind journalist once said of my automobile). We named it the ARC, an acronym for Average Response Computer [11].

The ARC was completed in early 1958. Though not all that much smaller than the TX-0, at least it was portable and rolled through doorways. I rode it in triumph into the Communications Biophysics Laboratory and it was almost immediately put into service. After about a year of operating experience the CBL staff decided that a third mode for compilation of time histograms of single neuron activity would be useful, and this was duly wired into the ARC control. Over the next several years the ARC served in a wide variety of studies, teaching researchers many new aspects of the neuroelectric behavior of the brain. It also confirmed my belief that there were indeed useful things that small digital computers could do in the laboratory.

But wouldn't it have been better to program these operating modes rather than wire them into the control circuits? I had since taken just that approach in designing an extremely simple stored-program computer of very limited capability, the 256 10-bit word L-1, for use in a special project [12]. There was no doubt in my mind that the far greater flexibility of the stored-program approach was of enormous value even in computers small enough to be considered "instruments" for laboratory use.

At the first symposium of the Brain Research Institute held at UCLA in 1960, Belmont and I joined with Walter and several members of CBL and Dr. M. A. B. Brazier (of both CBL and the BRI) in discussing some of the viewpoints and accomplishments of the M.I.T. work [13]. In my presentation I contrasted the stored-program and wired-logic approaches:

> The TX-0 is a relatively small but powerful stored-program computer. A rough comparison of the ARC and the TX-0 computers is illuminating. Both machines operate on 18-bit binary numbers, control analog-to-digital converters, have cathode ray tube displays, can be controlled by the experimenter, and hold about the same number of circuits derived from the same body of electronic technology. The TX-0, however, has about 8000 registers of digital storage and is organized as a general-purpose device.
>
> It is quite simple to program the TX-0 to act very much like the ARC; in fact this has been done, requiring an investment of effort measured only in hours. The TX-0 can in addition generate quite varied displays of the data and of results of analysis, and can be programmed to carry out exceedingly lengthy and complex operations if desired. This flexibility of behavior is characteristic of the stored-program computer and is obviously of great value.
>
> It is extremely important, of course, to provide for procedural flexibility and easy access to machines of this type in order to realize their full potential. Ideally the researcher would have the general-purpose computer in his laboratory for use "on-line", enabling him to observe and act on the basis of the calculated results while the experiment is in progress.
>
> Stored-program computers like the TX-0 are beginning to appear in commercial form and there is reason to hope that these machines, or perhaps other general-purpose machines with a capability somewhere between that of the ARC and the TX-0, will find their way into the laboratory.

Well, there it was. Clearly, the thing to do was to go off and design a machine that would fill the niche I had just defined.

DESIGNING THE LINC

It has been both pleasurable and dismaying to look over my design notebooks after a lapse of nearly twenty-five years. I can easily relive the the exhilaration and sense of discovery in the pages detailing those rare ideas that "worked" and survived and were good. But there are other pages, too, some of them recording in distressing detail my false starts, muddled thinking, and irredeemably bad ideas. The design process is like that, of course; yet one always hopes that the overall record will reveal a higher order of thought and development. It is embarrassing to report, therefore, that the LINC design notebooks do not, on balance, document the workings of an orderly mind.

Following the October symposium at UCLA I increasingly began to sketch and doodle on various handy surfaces. It seemed to me that I now had most of the keys to the design process: a firm belief in the soundness of the goals and a good sense of the functional requirements, the general technology to be used, and the bounds of acceptable size, complexity, and cost. What I was trying to find was some gimmick, some architectural idea to start the process off in any promising direction. Taking the principal constraint to be the factor of acceptable cost I arbitrarily set a target of

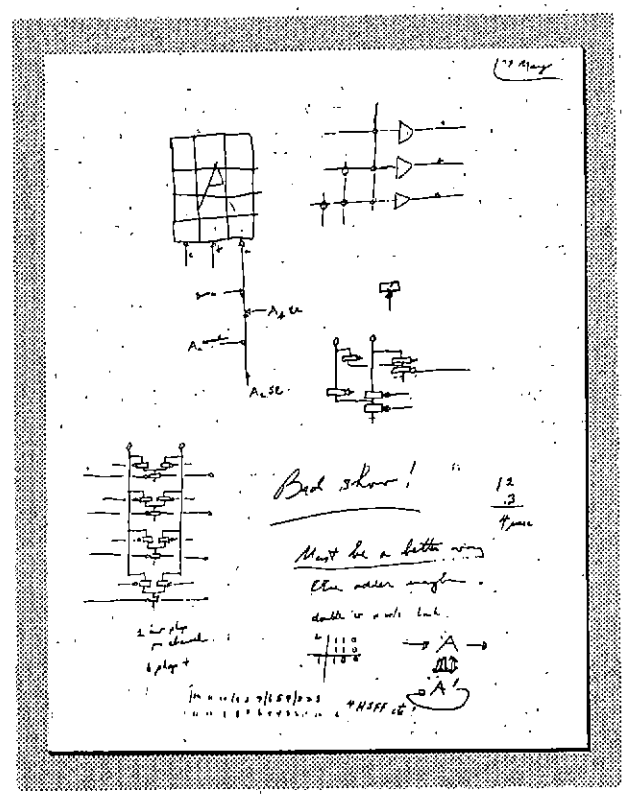
$25,000. This figure, relatively small for computers as we then knew them, seemed about right. It was about the size of one staff salary, for example, or about what a department head could authorize for equipment without the approval of higher management, and so on. Remarkably enough it was little more than twice the cost of the a-to-d converter used with the ARC. Was there some architectural scheme that might yield a useful laboratory-size computer subject to this cost constraint, something that would propel me out of the back-of-an-envelope phase and into the discipline of the engineering notebook?

First Attempts

Volume I of my three LINC notebooks carries the dates May 24 to July 4, 1961 — roughly Mother's Day to the Fourth of July. I had found a starting gimmick in the form of a serial-parallel scheme that appeared to reduce cost without seriously compromising speed. I would take advantage of a new family of plug-in circuit modules then being produced by the Digital Equipment Corporation. Ken Olsen and Harlan Andersen had left Lincoln Laboratory to found DEC in 1957 and were followed shortly thereafter by Dick Best, Ben Gurley (who later engineered the PDP-1), and others from the Lincoln group [3]. The circuit modules DEC was now manufacturing, forerunners of the integrated circuit chips that were not to appear for several more years, seemed quite suitable.

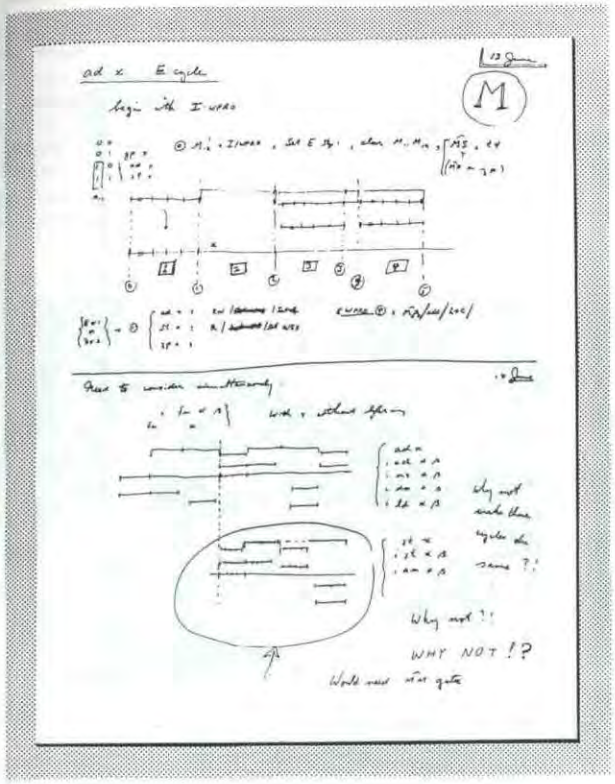
I knew that the machine would need two small tape units about the size of the "snapshot" unit I had proposed earlier for use with the TX-2 as a personal file input-output device and that my colleague Tom Stockebrand had tried to make. These units would have pocketably small reels for programs and data and would use block addressable magnetic tape like that of the TX-2's huge and frightening device for on-line files [14]. [The TX-2 tape unit used 14 inch reels of 3/4 inch tape, a very large inertial load. Its design was based on an observation of mine that the TX-2's multi-sequence capability made it possible to eliminate the usual requirement for finely controlled tape speeds, and I further proposed that the positions of all bit-cells on the tape be fixed by prerecording timing and blockmark tracks. In programming use, the resulting unit's tape speed varied markedly with wrap diameters, as did inertias and vibrational resonances. At top speed in block searching mode the tape's linear velocity reached 60 miles per hour! The whole room shook.]

Most of the work over these first two months went into thinking through the details of the serial-parallel architecture and an appropriate set of machine instructions. I knew that I had to build in a-to-d conversion channels and devote some of them to console potentiometers for convenient knob control of continuous variables (a good idea contributed by Belmont). I knew that CRT display would have to be inexpensive and would therefore use the same point-display technology that appeared in all of the M.I.T. machines. A more difficult task, however, would be thinking through the control for the tape units and developing the right set of block transfer modes for their use. The 1K or 2K 12-bit words of memory that seemed feasible under the cost constraint would not permit lengthy program routines for tape operation. These routines would therefore have to be wired into the central control circuits (of course there were no ROM chips in those early days). Block addressability of the tapes would help in running useful programs in the very small memory, the sort of thing we now do with disk systems.



The 27 May (1961) page from the LINC design notebooks. Early design was based on a serial-parallel logic using 4-bit shift register circuit modules made by DEC. Design progress was variably good.

I worked at home. Early in the design work I returned to the Laboratory briefly for an informal exposure of ideas to Charlie and Belmont and other members of the group, among whom were Severo Ornstein and a newcomer, Mary Allen Wilkes. There was a lively interest in what I presented as a "Linc". [I had thought that a generic name for this kind of computer would be useful and had decided on one that suggested the instrument's Lincoln Laboratory origins; Linc would be promoted to LINC by later events.] Following the meeting Severo and Mary Allen began to try programming with the proposed instruction set and I returned to the task of filling in design details. Charlie began to communicate his enthusiasm for the Linc scheme to CBL. Walter himself was later reported to be envisioning a relationship between a scientist and a computer similar to that between a scientist and a microscope. All was going well.



The 13 June (1961) page from the LINC design notebooks. Timing diagrams still appeared to support the serial-parallel scheme and new ideas were still emerging.

Alas, the serial-parallel approach turned out to be a false start. One month after its first exposure I had put enough of the design in place to establish the cost of the modules required by the central logic alone. Even without input-output devices or the core memory I had already used up my $25,000. Gad. The final entry in volume I is the single line: "Crisis in River City!"

The Prototype Linc

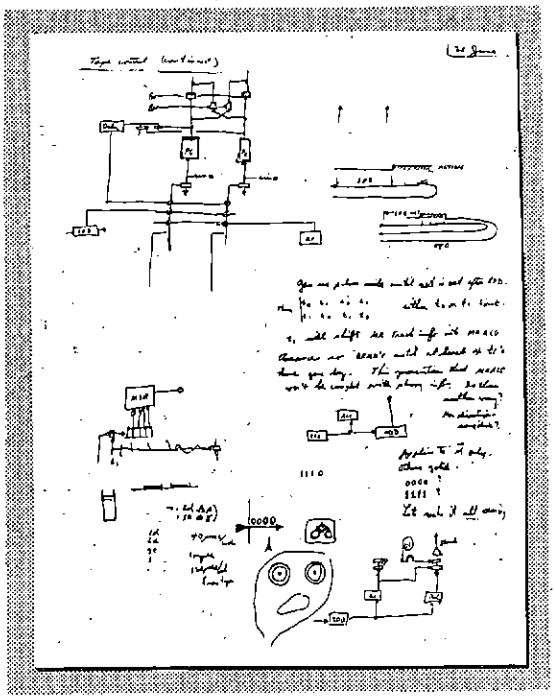
For the next several months I concentrated on simplifying the design and detailing the tape unit and its control, the most complex part of the machine and the largest single unknown. Much of my earlier work translated readily enough into a more straightforward 12-bit parallel form. Somewhat to my surprise this yielded a substantially lower cost. So much for gimmicks. Notebooks II and III record a great deal of work over the remainder of the year in refining the instruction set and developing simple operating modes for the tape units. Sketches for the tape transport mechanism and console appear here and there.

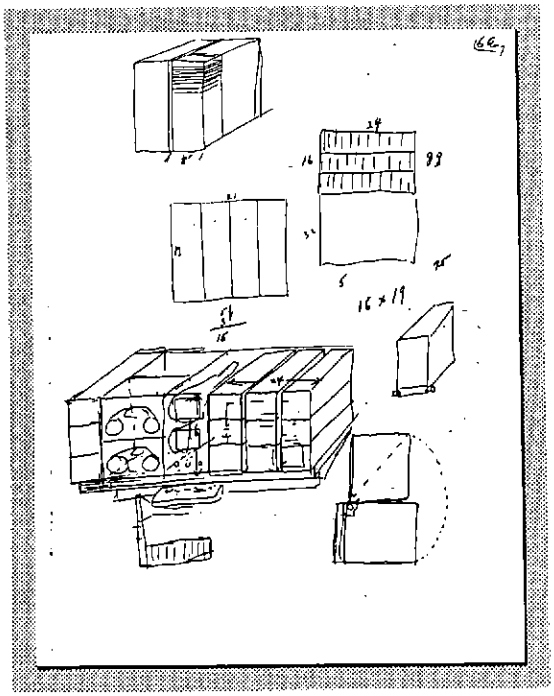
The need to build up a project effort at Lincoln began to demand more and more of my time as other members of the group joined in. Tom Stockebrand and I built a working model of the new tape unit, with Charlie contributing helpful insights from his experience with its fearsome progenitor, the giant tape unit of the TX-2. Severo designed and built a special subsystem to pre-mark tapes with the required fixed backbone of block addresses, timing signals, and control code patterns. Hershel Loomis and I assembled a tape unit exerciser/tester using DEC building block logic modules and established that the Linc tape equipment would indeed function as I had hoped. We were on the right track.

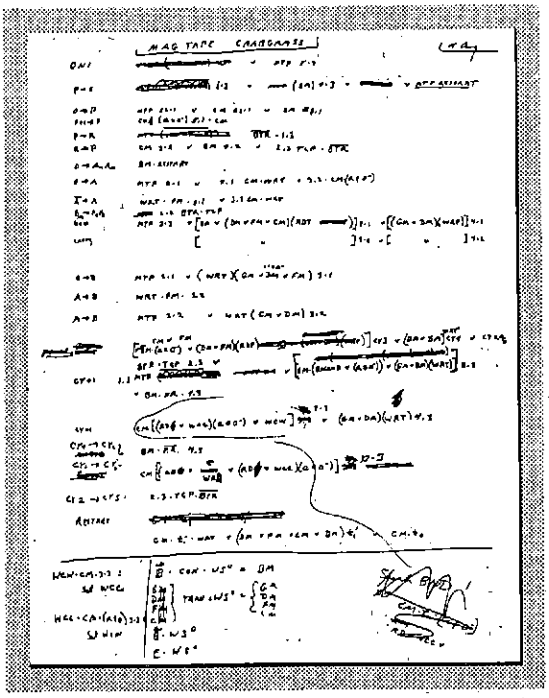
By early September the instruction timing diagrams had been completed in enough detail to enable Mary Allen to write a Linc system simulation program for the TX-2 and begin to develop a compact machine-code assembler for the proposed machine. Bill Simon, another newcomer and a remarkably creative scientist and engineer, designed a dual-scope alternative to the single-scope unit we eventually settled on, and began to write a number of test and demonstration programs. Norm Kinch, my right arm in matters of making things happen, began to produce large working drawings of system logic based on my notebook sketches and placed orders for parts as they emerged from the design. Severo and others helped with details of package layout and wiring lists. Charlie reviewed the design of the proposed 1K 12-bit memory system and watched over general engineering specifications. The final entry in Volume III was dated Christmas, 1961. Construction of the prototype Linc was already underway.

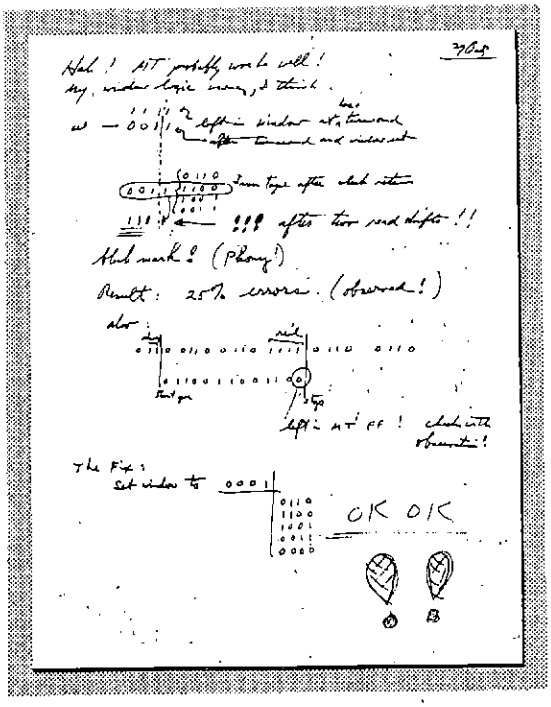
We hoped to complete the Linc prototype in time to take it to Washington for a demonstration in April at the forthcoming National Academy of Sciences Conference on Engineering and the Life Sciences. This had been urged by Walter and by Bill Papian as an excellent way to introduce the machine to a broad scientific and technical audience. Under this time schedule pressure but with tireless support from an enthusiastic technical staff, the Linc began to take shape. Hand-wiring of the electronics cabinet took its toll in anxiety and time over the ensuing weeks, as did the slow accumulation of special equipment and parts. But Norm kept the construction pretty much on schedule and the machine was completed in late February. It was working well enough for a Laboratory-wide demonstration at the end of March, 1962. The entire Linc group was justifiably proud of its accomplishment.

The demonstration prototype consisted of a set of four box-enclosed console modules, each connected by 20-ft long cables to a common electronics cabinet now the size of only *one* refrigerator; this general configuration would be used in all subsequent "academic" versions. One module, its box mostly empty, held a control panel that provided switches and register indicator lights (remember those?) together with speed and audio control knobs and so forth. A second module held a 5" CRT display adapted from a laboratory oscilloscope. A third module held the dual tape transport mechanics, while the fourth held a set of potentiometer knobs and jacks for analog input together with connectors for future input-output equipment. As an option for crowded laboratories, all modules could be removed from their boxes and mounted in standard equipment racks if desired. [Somewhat embarrassed









An abbreviated history of the LINC tape unit design. Upper left: June 28th sketches of parts of the tape turn-around control, including a picture of my foreboding of difficulties yet to come. Upper right: August 6th sketches of a possible arrangement of parts. Lower left: August 15th list of control logic expressions. Lower right: October 27th report of my realization that observed misbehavior of the prototype tape unit was caused by my design mistake. The unit worked well shortly thereafter. (From the 1961 LINC design notebooks).

by the size of the electronics cabinet, we adopted the point of view that it was only the console modules, taken together, that constituted the instrument itself; the rest was merely the electronics that made it all go and would be tucked out of sight in any convenient closet. We expected the cabinet eventually to disappear with advances in electronic packaging. It did, of course; but then so, eventually, did consoles.]



The Linc system demonstrated at the Lincoln Laboratory auditorium in March, 1962. Four console modules are connected by 20-foot cables to a cabinet holding the required digital electronics. The demonstration consisted of running a few simple programs toggled in by hand in advance.

The National Academy of Sciences Demonstration

Charlie, Norm, Bill Simon, and I took the Linc to Washington as planned and the National Academy demonstration was well if somewhat uncomprehendingly received [15]. Bill had arranged for closed circuit TV equipment and had set up several monitors about the meeting room. Charlie and I had spent the night before — *all* night — trying to fix some unexpected problem in the arithmetic element, crawling around the floor of our suite at the conference hotel with test equipment and soldering irons and discovering only by the dawn's early light that just outside our room there was a huge broadcast antenna tower that had been flooding everything with electromagnetic noise.

Norm commandeered one or two people from the hotel staff and moved the Linc to the meeting room for the presentation while Charlie and I dressed, Charlie in his Air Force uniform since at that time he was completing his military service at the Air Force Cambridge Research Laboratories and had to appear in offical garb. I stepped to the podium not knowing whether the machine was going to work or not, Charlie still frantically toggling in program parameters and trying out the demos. But at the last possible moment he slipped me a scrap of paper that said, "The following programs have my confidence: ... ". The Linc performed perfectly. Whew! The only question asked, Charlie recalls, came from an older gentleman who seemed to be quite interested in the durability of the wiring insulation. He was from the Smithsonian.

NIH Interest

After the Academy meeting Norm moved the Linc to one of the laboratories of Dr. Robert Livingston, Scientific Director of both the Institute of Neurological Diseases and Blindness and the Institute of Mental Health. There Charlie connected the analog-to-digital input channels to a multiple electrode array implanted in the brain of one of the lab's mascots, a cat whose name, I believe, was Jasper. In short order he wrote a small program that displayed on the CRT the average neuroelectric responses of the behaving animal. The Linc was successfully performing its first scientific task.

The writer Sam Rosenfeld, in a background paper prepared for a seminar held at NIH in celebration of the twentieth anniversary of the LINC, quotes Dr. Livingston's recollection of the event [16]:

> It was such a triumph that we danced a jig right there around the equipment. No human being had ever been able to see what we had just witnessed. It was as if we had an opportunity to ski down a virgin snow field of a previously undiscovered mountain.

[Jasper merely purred and looked pleased.]



Jasper, one of several animals used in experimental work at a National Institutes of Health laboratory. The other end of the cable leading from the cat's head was connected to the Linc in a 1962 trial, the Linc's first scientific assignment.

It must be reported, however, that we did not have much success the next day when the Linc was moved to the laboratory of Dr. Mones Berman to see what it could do to isolate a very fast fluorescence transient produced in a particular biochemical reaction. Severo and Mary Allen answered an emergency call and came to Washington to help, but to no avail. Bill Simon would later write a superb program for the general problem of exponential decomposition but the limits of Linc speed had been reached.

Nonetheless, the interest of NIH had clearly been engaged. The National Academy of Sciences had already made the suggestion to Dr. James Shannon, Director of NIH, that he set up a committee to monitor and encourage the development of computer technology relevant to biomedical research. In response, Dr. Shannon had established the Advisory Committee on Computers in Research. This committee, among whose members were both Dr. Brazier and Bill Papian, had been following the M.I.T. activities closely. The Linc demonstrations at NIH had in fact been arranged by the committee's executive secretary, Dr. Bruce Waxman, who would shortly take further initiatives leading to the funding of a Linc evaluation program.

Linc Becomes LINC

The group at Lincoln had been joined by Dr. Thomas Sandel, a neurophysiological psychologist who had been conducting auditory research at CBL. Tom had organized and hosted an NSF-sponsored workshop the previous summer in which some of the M.I.T. computing techniques were shown to an invited group of biomedical researchers. He now proposed to use the Linc in a repeat of the workshop to be held in the summer of 1962. Plans were approved and a well-attended workshop was held during which several new Linc programs were written and new applications were investigated. [An amusing side benefit: In programming one of the data recording tasks we discovered that we had neglected to wire in a pathway for the signal specifying the initial direction of tape motion. Since the tape units were designed to search for designated blocks in either direction (now at a sedate though still uncontrolled speed of about 3 miles per hour), the occasional misbehavior had been self-correcting and had largely gone unnoticed.]

We began to think in terms of expanding the work. Bill Papian had been given to believe that under the right circumstances funds for support of a larger program in biomedical computing would be forthcoming from NIH. Charlie and Bill Simon and I were already beginning to modify and improve the design of the Linc so that a number of soundly engineered replicas could be made, and Tom Sandel was proposing to set up a small lab in space adjacent to the TX-2 for relevant biologically oriented work. But it was not to be. Lincoln Laboratory management, sensing that there would be serious organizational difficulties in administering such a program within its established framework, firmly rejected both the expansion and "wet" lab proposals. Instead, we were invited to find a more suitable home for any further work. In reporting to a stunned Linc design team the management's decision and my own decision to leave, I announced that "Linc" had just become "LINC", an acronym for Laboratory INstrument Computer.

The principals subsequently spent several disheartening months trying without success to find a "suitable home" for the continuation and extension of the work to which they were now deeply committed.

THE LINC EVALUATION PROGRAM

Toward the end of 1962, Walter Rosenblith put forth an imaginative proposal: M.I.T. would act as host institution in a multi-institutional, multidisciplinary endeavor to be called The Center for Computer Technology and Research in the Biomedical Sciences; faculty and staff from several New England universities would join in providing the scientific and technical substance of a long-range program; a Center Development Office would immediately be set up to work out details, secure multi-institutional blessing, and formalize a proposal to NIH; and finally, to avoid loss of momentum the LINC team would immediately be taken in under the CDO wing to complete the redesign effort and mount a program for the dissemination of LINC technology.

This proposal was widely accepted almost at once. Funds for the establishment of the CDO, with Walter as Director and Bill Papian as Associate Director, magically appeared. Arrangements were made with Lincoln for the transfer of staff and equipment, and the entire biomedical research computer effort at Lincoln Laboratory, together with several members of CBL, moved to new quarters next to the campus in Cambridge in January, 1963.

Redesign of the LINC

The next several months were ones of intense activity as we worked hard to complete the redesign in time for what was now formally known as the LINC Evaluation Program. Construction of 16 LINCs was authorized under the formal program. Twelve of these machines were to be assigned to biomedical research scientists selected by a nationally constituted LINC Evaluation Board for trial use in their own laboratories following a special Phase I training program at M.I.T. Four machines were to remain with the design team for coordination of Phase II evaluation activities and further refinement of the instrument itself. Given the new CDO framework Dr. Waxman of the NIH Committee, in a dazzling display of civil service at its best, had been able to earmark about half of the needed funds from NIH and had persuaded Dr. Orr Reynolds of NASA to put up the remainder from NASA's recently established Bioscience Program. In parallel with this activity NIH assembled the necessary committees to review the much larger proposal for the Center itself, and with amazing smoothness and speed authorized a huge M.I.T. research grant of an unprecedented scale sufficient to warrant announcement on the front page of the New York Times.

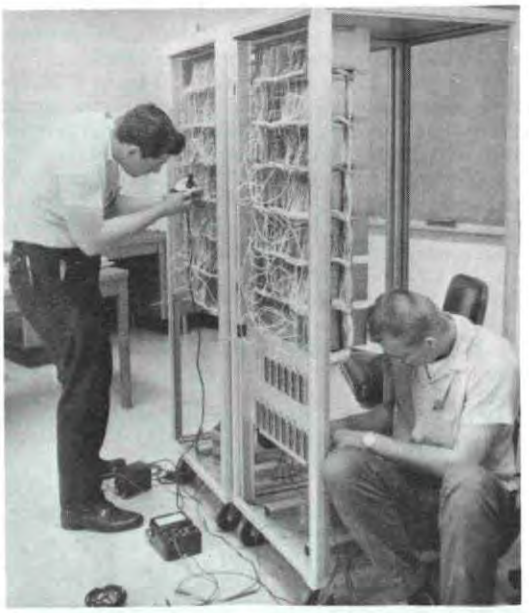
The idea of providing replicas of the LINC to individual researchers seemed quite natural. Tom Sandel had outlined just such a plan following the summer workshop the previous year and the idea had taken hold immediately. Charlie had then suggested, only partly in jest, that the computers be

put together by the workshop participants themselves and I had promptly taken the idea quite seriously as an excellent way to teach the inner workings of the LINC. After all, it would be up to the participants to keep their machines running, wouldn't it? What better way to ease these good people into the discipline of digital systems? We would gather the parts and subassemblies together in the form of a kit and provide assembly instructions and any necessary documentation. This would not only keep the costs down but would also encourage later participation by other interested individuals.



William Simon with an interim version of the LINC. The dual-display module provided both long- and short-persistence phosphor CRTs. Six potentiometer knobs supplied input parameters.

Again the projected time schedule was extremely tight, especially for a development group in an academic setting. But morale was very high as it always is under such circumstances and every member of the now somewhat expanded design team proceeded to put in a magnificent performance. Charlie designed new circuits for the memory (now modularized in two 1K units with only one to be included initially) and completely reworked the magnetic tape unit read/write amplifiers and motor control circuits. Bill Simon redesigned the a-to-d circuitry and worked out a new single-scope system for the display module. Severo expertly refined the tape control logic and incorporated the tape pre-marking subsystem. Mary Allen traveled back and forth to the TX-2 at Lincoln Laboratory to update the LINC simulator and complete the new LINC Assembly Program, LAP3. Don Malpass joined us temporarily from Lincoln and did an outstanding job of designing the power supply and the interconnection system, the high-speed computer's Achilles' heel if not very skillfully handled. Norm Kinch prepared and endlessly updated the necessary drawings, and supervised our support staff as well as the beginning manufacture of subassemblies by local vendors. I concentrated on the surprisingly difficult job of working out the details of console control and the proper arrangement of indicators and switches, and generally tried to keep the overall program on target.

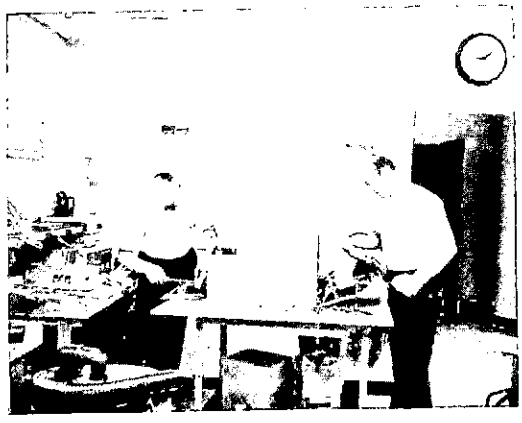


Technicians assembling wired frames for the LINC electronics cabinet. About twenty of these cabinets were required for the summer program.



Charlie debugging a prototype of the final LINC system. Modifications to the frame wiring were few but had to be made to all frames being assembled by a local vendor.

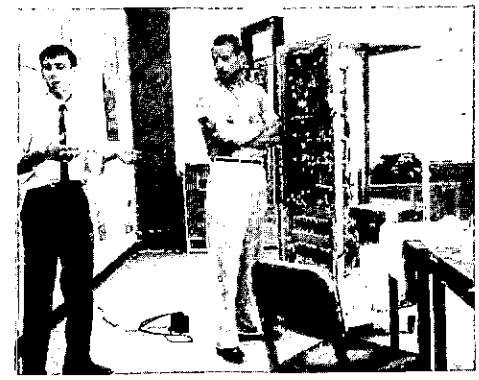
One month to go. Parts and subassemblies were continuing to arrive and the final LINC prototype neared completion. Mary Allen finished off the layout of the central control logic. Mishell Stucki, a man without peer in attention to detail, organized the wiring lists in a form suitable for the electronics frame checkout so critical to the success of the kit concept. Mort Ruderman of DEC took personal responsibility for assuring that sets of DEC modules, about 90% of the required electronics, would arrive on time. Howard Lewis and Dan Calileo and others stepped up the pace of special parts assembly under Norm and Charlie's supervision. Don O'Brien saw to it that the extensive and growing documentation was kept in good shape and had the assistance of Henry Littleboy of Massachusetts General Hospital. Severo was everywhere. One could not have hoped for a more dedicated and competent team.

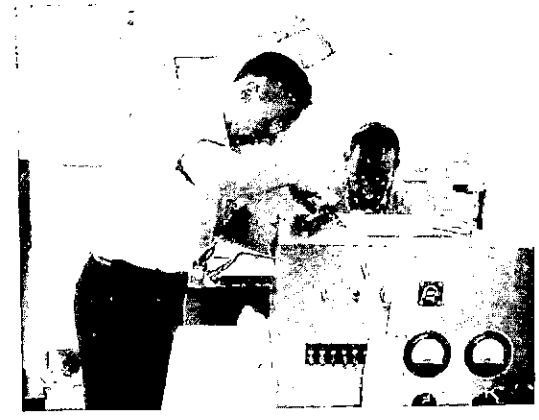




(Upper) Norm Kinch (left) and Dan Calileo repairing prototype power supply; (lower) Howard Lewis and Dan making "final" wiring changes to the electronics frames. Several additional modifications were later found to be necessary and were added, using white wires to avoid confusion with color-coded frame wiring already documented.

[Phase I photographs taken by Bill Simon.]







(Top to bottom) Severo Ornstein (left) and Prof. Jerome R. Cox of Washington University review last-minute problem; Bob Brown (left) and Tom Sandel examine incoming power-supply; I check the wiring of a control panel module.

One week to go. One or two frames had been completely wired but there was still difficulty in obtaining the necessary test equipment to verify wiring accuracy. A last minute circuit problem that Charlie had discovered was overwhelmed with the help of Maynard Engebretson and Prof. Jerome R. Cox (a member of the Evaluation Board), who made

a special trip from Washington University in St. Louis. Tom Sandel, Chairman of the LINC Evaluation Board, monitored progress with increasing anxiety. Charlie and I maintained a facade of manifest confidence that masked serious concern.

Phase I: The LINC Summer of '63

On July 1 the first group of visitors arrived, some of them with golf clubs they would never use. We immediately put them into a crash course on the theory and use of computers, a holding action managed by Mary Allen and Irving Thomae and Severo for the two more weeks it would be until the kits were ready.







(Top to bottom) Severo explains LINC logic to the first group of visiting scientists; Irving Thomae reviews timing diagrams; Mary Allen Wilkes shows LINC programming.





(Upper) Tom Sandel (right) in discussion with Dr. Gerhardt Werner of Johns Hopkins University; (lower) Mary Allen works on a program using rack-mounted equipment.

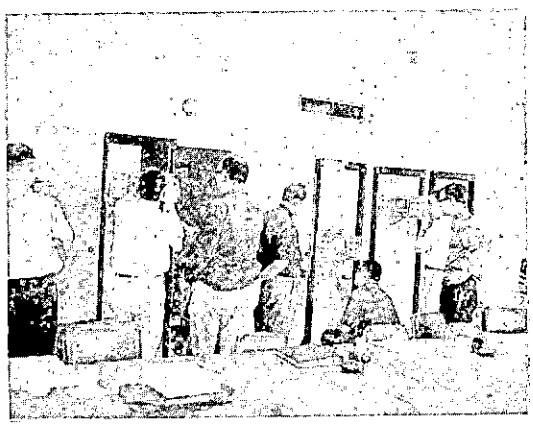
While classes were proceeding, the rest of us worked over the electronics frames and completed the job of checking the wiring, using Mishell's well-organized lists. The only part of this miserable and boring task we found rewarding was a sparkling Christmas-tree light effect produced by the test equipment. We found only a few wiring errors and fixed them.

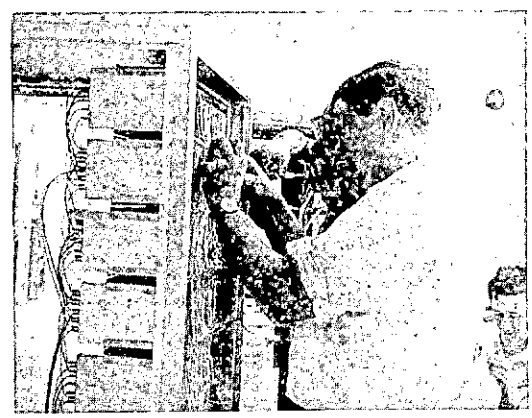
Finally the assembly process began. To our delight and relief all went smoothly, though many long days and nights of exhausting work were still required to make up the lost time and correct minor design mistakes as we found them. The visiting scientists energetically and with much spirit and enormous dedication to the task proceeded quite successfully to bring all six or so LINCs to life.







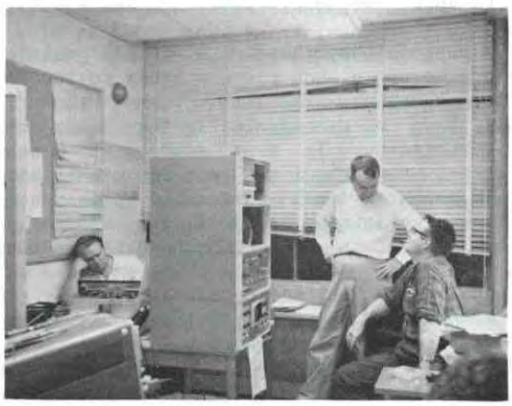




(Upper) Dr. A. J. Hance examines his tape unit module. Tape unit modules were almost entirely electromechanical; one critical component was a rubber-band. (Lower) A group of visiting scientists mount display units in staging racks.

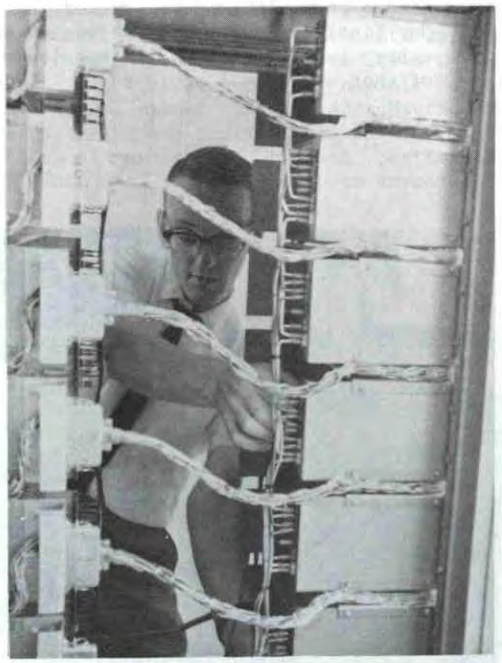
(Top to bottom) Dr. G. F. Poggio contemplates the plug-in unit installation task; Drs. N. Bell (left) and R. Stacy with staging rack; Dr. J. C. Lilly installs a modification while Mishell Stucki (rear right) ponders a problem.







(Top to Bottom) Dr. J. W. Woodbury installs a plug-in unit; Charlie discusses a point with Dr. K. Killam (right) while Dr. Hance nods (it is 2 a.m.); Drs. C. D. Geisler (left) and J. E. Hind examine wiring.





(Upper) Dr. C. A. Boneau reaches into his cabinet to make an adjustment. (Lower) Drs. F. S. Grodins (left) and J. E. Randall wrap interconnection cables.

Each of the visiting teams was required to write and demonstrate a small program representative of the work of its home laboratory. With the completion of these final assignments, the machines were packed up for shipment and the entire group of hosts and visitors, now fast friends, celebrated with great exuberance at a farewell banquet.

The M.I.T. hosts had only three days to recover before the second group of visitors arrived. This final session of Phase I went more smoothly than the first, with less pressured teaching and more time for thoughtful assembly. Once

again the visitors carried out all of their assignments with flair and great good humor, and again — on time — the LINCs were packed up for shipment. Our final banquet this time was an occasion for sober reflection on the magnitude and importance of what we had all achieved.



Moving day. Although some parts were crated for transportation, the electronics cabinets were simply taped up and carried in moving vans as furniture. Most of the machines were moved in this manner and all arrived safely at their new homes in laboratories across the country.

The LINC was now well and safely launched and in good hands. Charlie and I had little doubt of its ultimate success.

For the most part the participants had all taken to the unfamiliar fairly well. For those few who had already worked with computers, the general acceptance of machine-level programming as the price of dealing with such a small memory had been cheerful enough. Productive software, however, would not begin to accumulate in the various laboratories until the following year; the first published version of the programming manual, the year after that; and the first edition of the final assembly program, LAP6, not until 1967.

We had had to formulate both teaching and assembly procedures as we went along, but this resulted in only a few bruised knuckles and egos and a few inefficient hours (one of the visitors, a neurosurgeon, did have difficulty adjusting a tiny slotted-screwhead potentiometer deep within the recesses of the electronics frame, as I recall). Fixing design mistakes and modifying the circuits and wiring to reduce system noise meant repairing all machines in parallel. Each machine, by the way, had required a budget outlay of about $32,000, or so the accountants told us. Not too bad a miss.

Throughout it all we had been sustained in our endeavors not only by our boundless energy and enthusiasm, our spirited sense of adventure, and our deep reserves of unshakeable conviction and commitment, but also by the Fox & Tishman Restaurant of Kendall Square, Cambridge. The F&T's sympathetic management and colorful staff fed and otherwise refreshed us at all hours of the day and night. [Following a suggestion of Charlie's, any member of the team who was responsible for a design goof agreed to buy one martini for all other members of the team at the end of the effort — one martini per goof, non-cancelling. On the day of reckoning we all repaired, of course, to the F&T. I can't remember how it all turned out.]

The Move to Washington University

The weeks immediately following our intensive summer effort did not go well. We were not only exhausted but were also confronted with an unanticipated organizational problem. The nascent M.I.T. Center, prospectively multi-institutional and multi-disciplinary, turned out to be irremediably multi-problematical as well and de-materialized. The result was that once again the peripatetic LINC team found itself in need of a more suitable home for its work. We were ...um... disappointed.

I suppose that these days it is natural to expect a group in such circumstances to form a commercial company. The LINC team, however, had always been more academically than entrepreneurially inclined. In many subsequent months of searching, the principals traveled about the country and met with many university presidents and trustees, examining several propositions quite seriously. But it was an unscheduled meeting in Cambridge with George Pake that convinced us that the best choice for the continuation of the program was Washington University in St. Louis, where Dr. Pake was Provost in the years before he left the academic world to set up XEROX PARC. He had heard about our situation from Professor Cox, he said, and had "just happened" to be visiting Cambridge on his way to Woods Hole (he never got there). He had stopped by so that we could, in his words, look him over. We were already well acquainted with Jerry Cox, an old friend even then, especially since those final hectic days before summer, and had a high regard for his innovative work in biomedical computing at Washington University. For many of us, then, this extraordinary meeting with George Pake made an already very promising possibility irresistible.

And so it was that the Washington University Computer Research Laboratory was established in 1964 under the direction of William N. Papian, Associate Dean of Engineering. CRL set up shop in newly renovated space made available by its newfound "sister laboratory", the Biomedical Computer Laboratory under Jerry Cox. Tom Sandel accepted an appointment as Professor of Psychology, and I became Research Professor of Computer Science and Associate Director of CRL. Many from the Cambridge team, including Severo, Mishell, Howard and Ken Lewis, and Constance and Joe Towler uprooted and moved to St. Louis, followed by others from M.I.T. over the next few years. Charlie remained in Cambridge to complete both his military service and his doctorate at M.I.T. and rejoined the group a

year later as Associate Professor of Physiology and Biophysics. Norm Kinch rejoined us after an assignment at an experimental radar site operated by Lincoln Laboratory in the Southwest. Mary Allen received her appointment letter from George Pake midway in a one-year trip around the world (she was then in Calcutta), but before coming to St. Louis spent another year as an extramural member of CRL working on a new assembly-program/operating system, LAP4, using a LINC set up in the living room of her family home in Baltimore — surely a first in personal computing. But the original team never entirely regrouped. To our great regret, George Gerstein and Belmont decided that their immediate scientific interests were better served by accepting appointments at the University of Pennsylvania in the Laboratory of Dr. Britton Chance, and Bill Simon elected to accept an appointment at the Harvard Medical School.

Phase II Activities

Four main tasks faced us in carrying out our Phase II responsibilities under the formal evaluation program.

1) We had to assure that the LINC equipment itself had been transplanted successfully into each of the participants' laboratories and was kept up-to-date.

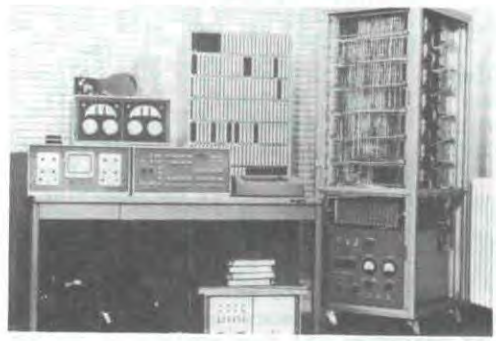
2) We had to complete and distribute further documentation on the machine and its use.

3) We had to improve LINC software, starting with an enhanced version of the basic assembly program that provided automatic filing capabilities and improved editing and operational features.

4) Finally, we had to accumulate operating experience with the LINC in our own work in biomedical research and in other application areas.

Assuring ourselves and our sponsors that the LINC was taking root required considerable use of the telephone and post office as well as visits to each site. A few faulty switches and parts needed to be replaced, a few adjustments made, and a few circuit improvements installed as we verified their need. We adhered to the principle that any modification to one machine meant a modification to all machines. By and large, however, machine reliability was excellent, thanks to sound engineering by Charlie, Bill Simon, and Don Malpass as well as to the robustness of DEC circuit packages when carefully used. [From one of the sites, the only call for hardware help we ever got dealt with what to do about the fact that the elapsed-time meter on the power supply had just jammed at 99999.]

Completing a set of finished documents on LINC hardware and operation was more difficult (the evaluation program participants had been able to take away from Cambridge only the roughest of descriptive material and operating notes). Among other things, it meant that a full production version of a LINC kit and its accompanying assembly and test procedure documentation had to be checked out and certified. Not long after CRL had been set up, Mort Ruderman brought the first complete DEC-produced kit to St. Louis. Mishell and Severo then put it together and verified the accuracy of the documents and test procedures, making a few necessary corrections. Final documents were prepared with the help of newcomers Dave Stewart (who had worked with Charlie during his Air Force days), Gerald Johns, Robert Ellis, and Maurice Pepper. After printing and distributing revised final documents to all evaluation program participants, we sold document sets at cost to anyone who wanted them.







(Top to bottom) The LINC kit produced by DEC; Mort Ruderman of DEC (left), Mishell (rear) and Severo assembling the kit; Norm Kinch stacks console modules.

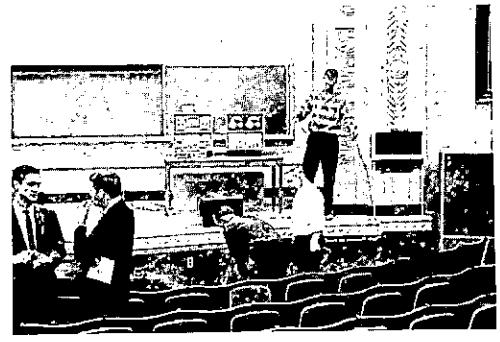
Building up a base of operating software was accelerated with the help of the Biomedical Computer Laboratory. A variety of utility programs were written by our BCL colleagues Jerry Cox, Mike MacDonald, Sharon Davisson, and others. BCL incorporated these into LAP4 (an interim modification of the LINC Assembly Program, LAP3), and many of its operating features were later incorporated by Mary Allen in the ultimate version, LAP6. Software was generally distributed on LINC tape reels via the mails. Since direct exchange of programs among the participants was rare, we did what we could to certify and distribute copies of application software that was submitted to CRL or written at CRL or its sister laboratory BCL.

Use of the LINC in our own work was supplemented by work at BCL, where a few kits were assembled and used in connection with collaborative programs with the Washington University School of Medicine and its associated hospitals. Later we acquired several of the LINC variant machines made by DEC and others and began to use them in our developing research program in macromodular systems (which soon took over as the major focus of CRL activities). [BCL went on to design the Programmed Console, the result of a design seminar that Jerry and I taught in 1965 [17]. Known as the PC, it provided both autonomous and remote-connnection modes of operation and thus was one of the earliest "smart terminals". It was subsequently produced in small numbers and made available to radiologists for radiation treatment planning under a national program modeled after the LINC Evaluation Program.]

The Final Evaluation Program Meeting

The program participants began to use and then gradually depend on the LINC in their research. It had not seemed so, however, at our first regathering in June, 1964, when we all met for a show-and-tell at a distinguished old resort hotel in New Hampshire. Just short of a year since the Cambridge Summer, there hadn't yet been enough experience gained in the use of the new tool, especially in view of the scarcity of software and the upsetting problem of relocating the LINC core group. No one was greatly surprised. But at the final LINC Evaluation Program meeting in St. Louis in March, 1965, the participants were well prepared and had by then accomplished a great deal of scientific work. The studies reported covered a wide range of topics [18]. Some of them were:

> Operant conditioning of pigeons and monkeys. [One series of experiments made use of a wonderful Lazy Susan holding a pigeon at each of four stations around its circumference, with indexing into data-collecting position, stimulus generation, and data analysis all provided by the LINC.]
>
> Spontaneous activity in thalamic and cortical neurons, behavior of single cochlear neurons, and somatosensory responses evoked in intracranial stimulation of cortex.
>
> Pulsatile blood flow, cardiac muscle behavior, hydrodynamics of the mammalian arterial system, and human finger tremor.
>
> Genetics at the bacterial and molecular level using mass spectrometers and other instruments.
>
> Communication between man and dolphin. [An unanticipated equipment hazard: the dolphin, one Elvar, could accurately spit a stream of seawater a distance of twenty feet.]







Preparation for the Final LINC Evaluation Program Meeting, March, 1965. (Top to bottom) I talk with Joseph Foley of Spear, Inc. while a LINC is being set up on stage; Prof. Harold Shipton (left) in discussion with Mort Ruderman and Winn Hindle of DEC; Drs. L. Wienckowski (left) and Bruce Waxman of NIH.

In addition to their scientific accomplishments the participants reported several other technical and software development activities. These included the connection of the LINC to computation center machines, the development of a system for remote control of the LINC, the design of interface circuits for control of a blue-format tape transport and other laboratory equipment, and the development of several experimental operating system programs and floating-point routines.

By the end of Phase II the Evaluation Program participants had an accumulated experience of about 50,000 hours of LINC use with remarkably little trouble, and at least that many hours had also been accumulated at other LINC sites with similar results. For the most part the original participants were awarded permanent custody of their machines following their reports of this strikingly successful operation in laboratories across the country.





(Upper) Tom Sandel and a now bearded Severo in a visit to the laboratory of Dr. G. S. Malindzak; visits were made by various members of the Washington University group to each of the participating Evaluation Program laboratories. (Lower) The author and William N. Papian with the final version of the LINC [Photo courtesy of the Saint Louis Post-Dispatch].

IN LATER YEARS

Although our formal responsibilities under the LINC Evaluation Program had been discharged, there was still work to be done that would take some of us a few more years.

Final LINC Modifications

Following the recommendation of the growing band of LINC users as reported at the final Evaluation Program meeting in St. Louis, a decision was made to undertake a number of hardware modifications. A new register and an arithmetic overflow circuit were provided to extend the range of arithmetic and to facilitate running a certified floating-point software package for which a considerable consensus had developed among the mathematically inclined. These changes were seasoned, documented, and distributed (with a few installation visits) to all of the original LINC sites, as was the floating-point package itself. Similarly, data and program interrupt features were incorporated to improve performance in several high rate applications. Altogether six new instructions dealing with these new features were wired into all machines.

The Final LINC Operating System

The final assembler/operating system, LAP6, was put together by Mary Allen with great care. Discussion of its specification had begun early in 1965 during her visits to several of the participating laboratories to develop a consensus regarding users' needs. A decision to double the initial 1K memory size had already been made; but as Mary Allen put it, doubling a 1K memory produces another small memory. Despite this limitation she was able to define a remarkably efficient operating system based on a LINC tape scrolling algorithm developed by Mishell and Severo. LAP6 included a line-by-line editor, an assembler, automatic management of files, user-defined metacommands, and runtime debugging aids [19]. It was thoroughly tested before being distributed in 1967 and the accompanying LAP6 Handbook still stands as a model of simplicity, accuracy, and usability [20]. A student of philosophy (today a lawyer in Massachusetts), Mary Allen quoted Kierkegaard in the foreword to the handbook:

To promise the System is a serious thing.

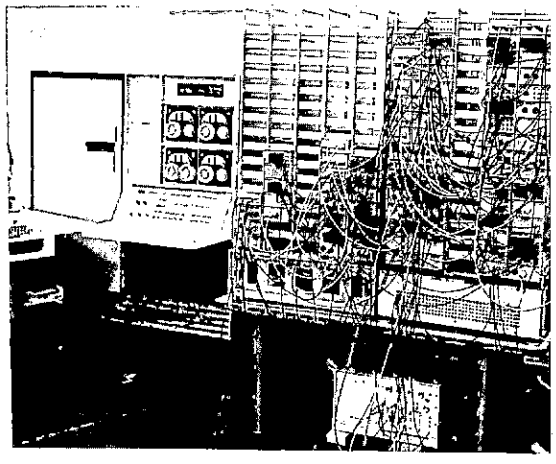
The handbook and the second edition of its companion programming tutorial document published in 1969 reflect the final form of what is now referred to as the "classic" LINC [21].

Manufacture of the LINC and LINC Variants

DEC went on to manufacture altogether fifty or sixty classic LINCs, only a few of these in kit form — an idea whose time never quite came. [Some of the kits were assembled for others by Prof. Harold Shipton of the State University of Iowa, previously a member of the LINC Evaluation Board. "Shippy", one of the most charming people ever to walk the face of the planet and well known to the biomedical community for his perceptive and imaginative engineering, later joined the faculty of Washington University.]

In 1965 I joined forces with Dick Clayton of DEC to design the LINC-8, a 32K dual-machine combination of the LINC and Gordon Bell's PDP-8, in which the complex LINC tape control logic was relegated to a concurrently executable program on the PDP-8 side using software written by Severo. While the nearly 150 LINC-8s were being made, Dick designed a new version of the machine, the PDP-12, of which about 1000 were manufactured [3].

In 1965 a new Massachusetts start-up company, Spear Inc., made its appearance. It began to manufacture a 4K integrated circuit version of the LINC, called the micro-LINC 1. Spear went on to produce the micro-LINC 300, a higher-performance 32K variant. [Spear, Inc. subsequently became a division of Becton Dickinson and Company. I have no doubt that the transaction helped Spear's former president farther along toward his life-long goal of retiring to some small South Sea Island that had an 18-hole golf course and a native monarch.]



A Spear Inc. micro-LINC 300 (left) providing operational support to an experimental macromodular system at Washington University.

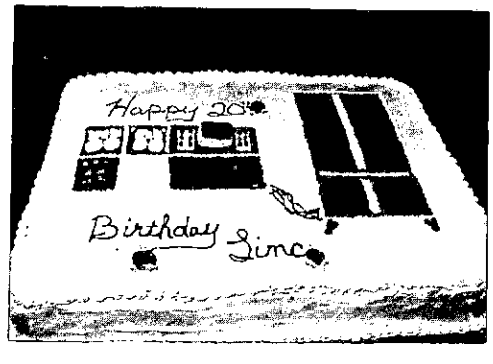
DENOUEMENT

With more than 1200 LINC or LINC variants at work around the world, research over the ensuing years began to result in very substantial contributions to the body of scientific literature. An effort was made to maintain a bibliography of LINC-related publications for the first few years, but the task soon got out of hand. The July, 1969 edition lists more than 150 scientific papers, journal articles, and published books [22]. By then the growth rate of bibliographic references was already doubling each year, and the number of LINC programs and application notes alone soon became so large that responsibility for handling them had to be turned over to the DECUS Program Library.

The careers and lives of those associated with the LINC development continued to grow and change in many ways, as new academic appointments were assumed and new professional challenges undertaken. Many early students of these scientists and engineers, and their students in turn, went forth to seek their fortunes after training in the LINC and its ways.

Washington University's program in the medical and biological research uses of computer science and technology grew in prominence and soon came to be regarded among the foremost in the country. In the course of years Prof. Jerome R. Cox became Chairman of the Department of Computer Science, and Prof. Charles E. Molnar, Director of the Institute of Biomedical Computing. Dean William N. Papian retired from the University to private consulting practice, as did I. In a very saddening turn of the historical wheel Prof. Thomas T. Sandel, after many years as Chairman of the Washington University Department of Psychology, died just weeks before the LINC's twentieth anniversary gathering of old friends of the Summer of '63 was to take place.

The LINC design went on to influence the design of DEC machines that further reinforced the concept of the small computer as a tool for personal work and very soon took over center stage. Yet over the years the LINC and its variants continued to serve the needs of biomedical research. A few of them were still on the job productively in 1985, and one of these — perhaps the last of the "classics" — could be found at the Massachusetts General Hospital, not yet retired after more than two decades of continuous use.



The birthday cake that made its appearance at the LINC Twentieth Anniversary Celebration held on November 30, 1983 at NIH.

References

[1] Clark, W. A. and C. E. Molnar: A Description of the LINC. in *Computers in Biomedical Research I*. R. W. Stacy and B. D. Waxman (eds) Academic Press, New York, 1965; ch 2.

[2] Redmond, K. C. and T. M. Smith: *Whirlwind - The History of a Pioneer Computer*. Digital Press, Bedford, Mass., 1980.

[3] Bell, C. G., J. C. Mudge, and J. E. McNamara: *Computer Engineering - A DEC View of Hardware Systems Design*. Digital Press, Bedford, Mass., 1978.

[4] Farley, B. G. and W. A. Clark: Activity in networks of Neuron-like Elements. in *Information Theory, Fourth London Symposium*. C. Cherry (ed), Buttersworth, London, 1961: 242-251.

[5] Clark, W. A.: Design Considerations for an Experimental Computer. Linc. Lab. memo 6M-3536, April 1955.

[6] Clark, W. A.: The Multi-Sequence Concept. Linc. Lab. memo 6M-3144, November 1954.

[7] Forgie, J. W.: The Lincoln TX-2 Input-Output System. *Proc. 1957 WJCC*: 156-160.

[8] Thacker, C. P. *et al.*: ALTO: A Personal Computer. in *Computer Systems: Principles and Examples*. Siewiorek, Bell, and Newell, (eds), McGraw Hill, 1982; ch 33.

[9] Mitchell, J. L. and K. H. Olsen: TX-0, A Transistor Computer with a 256 by 256 Memory. *Proc. 1956 EJCC*. 10: 93-101.

[10] Clark, W. A.: The Lincoln TX-2 Computer Development. *Proc. 1957 WJCC* :143-145.

[11] Clark, W. A., R. M. Brown, M. H. Goldstein, C. E. Molnar, D. F. O'Brien, and H. E. Zieman: The Average Response Computer (ARC): A Digital Device for Computing Averages and Amplitude and Time Histograms of Electrophysiological Responses. *IRE Trans. Biomed. Electronics*, 1961, BME-8:46-51.

[12] Clark, W. A.: A Functional Description of the L-1 Computer. Linc. Lab. Rept. 51G0012, ASTIA 236678, 1960.

[13] Clark, W. A.: Digital Techniques in Neuroelectric Data Processing. in *Computer Techniques in EEG Analysis*, M. A. B. Brazier, ed. EEG J. supp. 20, Elsevier, 1961.

[14] Best, R. L. and T. C. Stockebrand: A Computer-Integrated Rapid-Access Magnetic Tape System with Fixed Address. *Proc. 1958 WJCC*.

[15] Clark, W. A.: The General Purpose Computer in the Life Sciences Laboratory. in *Engineering and the Life Sciences*. NAS-NRC Rept., Wash. D.C., April 1962; 33ff.

[16] Rosenfeld, S.: LINC - The Genesis of a Technological Revolution. *LINC 20th Anniversary Seminar*. NIH Rept. November 30, 1983.

[17] Cox, J. R.: A Description of the Programmed Console. BCL Monograph #37, Wash. Univ., September 1966

[18] *Convocation on the Mississippi: Final Report of the LINC Evaluation Program*. Wash. Univ., 1965.

[19] Wilkes, M. A.: Conversational Access to a 2048 Word Machine. *Comm ACM*. 13:July 1970; 407-414.

[20] Wilkes, M. A.: *LAP6 Handbook*. CRL Tech. Rept. #2, Wash. Univ., May 1967.

[21] Wilkes, M. A. and W. A. Clark: *Programming the LINC*. Wash. Univ., January 1969.

[22] Wilkes, M. A.: Bibliography of LINC-Related Publications. CSL Ref. Lib. LINC Doc. #14, Wash. Univ., January 1969.

APPENDIX

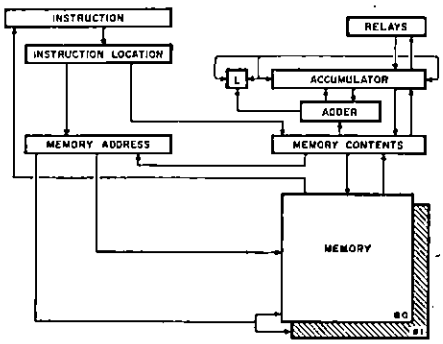
The LINC Evaluation Program Participants

Without any doubt whatsoever, as Tom Sandel would have said, the success of the LINC is due in incalculably large part to the pioneering work of the evaluation program participants. Its formal members were: Drs. E. O. Attinger and A. Anne, Research Institute, Philadelphia Presbyterian Hospital; Dr. D. S. Blough, Dept. of Psychology, Brown University; Drs. S. Goldring and J. L. O'Leary, Dept. of Neurology, Washington University; Drs. F. S. Grodins and J. E. Randall, Dept. of Physiology, Northwestern University; Drs. J. E. Hind and C. D. Geisler, Laboratory of Neurophysiology, University of Wisconsin; Dr. J. Lederberg and Lee Hundley, Dept. of Genetics, Stanford University; Dr. C. A. Boneau, Dept. of Psychology, Duke University; Dr. J. C. Lilly, Communication Research Institute; Drs. G. S. Malindzak and F. S. Thurstone, Dept. of Physiology, Bowman-Gray School of Medicine; Drs. G. F. Poggio, G. Werner, and V. B. Mountcastle, Dept. of Physiology, and Dr. B. Weiss, Dept. of Pharmacology and Experimental Therapeutics, Johns Hopkins University; and Drs. J. W. Woodbury and A. M. Gordon, Dept. of Physiology, University of Washington.

The number of LINCs actually built in the summer of 1963 was about twenty. Several other investigators had been invited to participate on an informal basis. These informal participants assembled their computers alongside the others, using their own funds but taking advantage of whatever batch orders were placed for parts and subassemblies. They were: Drs. K. Killam and A. J. Hance, Dept. of Pharmacology, Stanford University; Dr. H. E. Tompkins and Mr. J. S. Bryan, National Institutes of Health; Dr. J. B. Lewis, M.I.T. Lincoln Laboratory; and Drs. R. Stacy and N. Bell, Institute of Statistics, North Carolina State College.

Under the chairmanship of Dr. T. T. Sandel of

M.I.T., the LINC Evaluation Board included Dr. M. A. B. Brazier of the University of California at Los Angeles, Dr. J. R. Cox of Washington University, Dr. E. R. Dempster of the University of California, Dr. J. Macy, Jr., of the Albert Einstein School of Medicine, Dr. H. V. Pipberger of the Veterans Administration, Dr. M. D. Rosenberg of the Rockefeller Institute, and Prof. H. W. Shipton of the State University of Iowa.



THE EVALUATION PROGRAM LINC INSTRUCTION SET

FULL ADDRESS CLASS — ADD | X

The right-most ten bits of the instruction word specify one of the memory registers 0–1777 (octal).

ADD X	Add the contents of memory register X to the Accumulator.
STC X	Store the contents of the Accumulator in memory register X, and clear the Accumulator.
JMP X	Take the next instruction from memory register X, and save the subroutine return point in memory register 0.

INDEX CLASS — LDA | i | β ; LDA | i | 0 ; OPERAND OR ADDRESS

If $\beta \neq 0$, the instruction is single word length and the address of the memory operand is located in index register β ($1 \leq \beta \leq 17$ octal). If $\beta = 0$, the instruction is double word length and the second word is either the operand (case $i = 1$) or the address of operand (case $i = 0$). The specified index register is incremented by ONE before use if $i = 1$.

LDA i β	Load the Accumulator with the memory operand.
STA i β	Store the contents of the Accumulator in the specified memory location.
ADA i β	Add the memory operand to the contents of the Accumulator.
ADM i β	Add the memory operand to the contents of the Accumulator and leave the sum in the Accumulator and in the memory operand location.
LAM i β	Add the memory operand to the contents of the Accumulator, together with any previous overflow held in the Link Bit. Leave the sum in the Accumulator and the memory operand location, and retain any new overflow in the Link Bit.
MUL i β	Multiply the contents of the Accumulator by the memory operand, and retain either high-order half or low-order half of the double-length product in the Accumulator.
SAE i β	Compare the contents of the Accumulator with the memory operand, and skip the next instruction if they are identical.
SRO i β	Rotate the memory operand to the right one place and skip the next instruction if the sign bit of the operand is ZERO.
BCL i β	Clear each bit in the Accumulator which corresponds to a ONE in the memory operand.
BSE i β	Set to ONE each bit in the Accumulator which corresponds to a ONE in the memory operand.
BCO i β	Complement each bit in the Accumulator which corresponds to a ONE in the memory operand.
DSC i β	Display character intensifying points on the scope in a 2 by 6 array according to the bit pattern of the memory operand.

HALF-WORD CLASS — LDH | i | β ; LDH | i | 0 ; OPERAND OR ADDRESS

Addressing is similar to that of the INDEX CLASS, except that the left-most bit of the address specifies which half of the memory operand is used. When $i = 1$ and $1 \leq \beta \leq 17$ (octal), the address word in index register β is incremented before use, in such a way as to step through consecutive half-words, the address of the operand increasing by one on every second reference.

LDH i β	Load the Accumulator with the specified half of the memory operand.
STH i β	Replace the specified half of the memory operand with the contents of the right half of the Accumulator.
SHD i β	If the contents of the right half of the Accumulator are not identical to the specified half of the memory operand, skip the following instruction.

TAPE CLASS — (MTP) | i | u | RDC ; SECTOR | BLOCK NUMBER

These are double word length instructions in which the first word specifies the tape unit (u-bit), the motion state following execution (if $i = 0$, the selected unit is stopped), and in which the second word specifies one of eight sectors of memory and one of 512 tape blocks between which transfers are to be made. All required searching operations and checking are performed automatically, and are repeated if checks indicate improper transfer.

RDC i u Q, N	Read tape block N into memory sector Q, and check the transfer.
RCG i u Q, N	Read from one to eight consecutive tape blocks, beginning with block N, into memory and check all transfers.
RDE i u Q, N	Read tape block N into memory sector Q.
MTB i u Q, N	Move toward tape block N on specified unit.
WRC i u Q, N	Write contents of memory sector Q in tape block N and check the transfer.
WCG i u Q, N	Write from one to eight consecutive memory sectors on tape, beginning with block N, and check all transfers.
WRI i u Q, N	Write contents of memory sector Q in tape block N.
CHK i u Q, N	Find tape block N and leave the check sum for this block N in the Accumulator.

SHIFT CLASS | ROL | i | n

If $i = 1$, the Link Bit is coupled into the shift path. $0 \leq n \leq 17$ (octal)

ROL i n	Rotate the contents of the Accumulator n places to the left.
ROR i n	Rotate the contents of the Accumulator n places to the right.
SCR i n	Multiply the contents of the Accumulator by 2^{-n}.

SKIP CLASS | SNS | i | n

Skip the next instruction if the specified conditions are met. If $i = 1$, the skip conditions are reversed.

SNS i n	Check whether Sense Switch n is up, $0 \leq n \leq 5$.
AZE i	Check whether the Accumulator contains ZERO.
APO i	Check whether the contents of the Accumulator are positive.
IBZ i	Check whether either tape unit is reading an interblock zone mark.
SXL i n	Check whether External Level input line n is negative, $0 \leq n \leq 14$ (octal).
KST i	Check whether a key has been struck on the keyboard.
LZE i	Check whether the Link Bit contains ZERO.

SET | SET | i | β

OPERAND OR ADDRESS

SET i β	Set register β, $0 \leq \beta \leq 17$ (octal), to the value of the second word (case $i = 1$) or to the value of the operand whose address is located in the second word (case $i = 0$).

DISPLAY | DIS | i | β

DIS i β	Intensify a point on the scope whose vertical position is given by the right-most nine bits of the Accumulator and whose horizontal position by the right-most nine bits of memory register β, $0 \leq \beta \leq 17$ (octal). If $i = 1$, increment contents of register β before use.

INDEX AND SKIP | XSK | i | β

XSK i β	Increment by 1 the contents of memory register β (if $i = 1$) and skip the next instruction if the new contents of this register equal 1777, 3777, 5777, or 7777 (octal).

SAMPLE | SAM | i | n

SAM i n	Convert the analog signal appearing on channel n, $0 \leq n \leq 17$ (octal), to an eight-bit binary number in the Accumulator. If $i = 0$, this instruction takes 24 microseconds; if $i = 1$, the instruction is completed in 8 microseconds although the conversion process continues for an additional 16 microseconds.

MISCELLANEOUS | HLT

HLT	Halt the computer and sound a gong.
CLR	Clear the Accumulator and Link Bit.
ATR	Load the six Relay flip flops from the right half of the Accumulator.
RTA	Load the Accumulator from the six Relay flip flops.
NOP	Do nothing for 8 microseconds.
COM	Complement the Accumulator.

OPERATE CLASS | OPR | i | n

These instructions form a general input-output command set whose execution is partially controlled by externally generated signals. Some functions of this instruction set are summarized below.

1. *Pausing*, with conditional restarting.
2. *Generating outputs* on any one of sixteen control output lines.
3. *Reading into the Accumulator* from either of two twelve-bit parallel sets of gates.
4. *Reading into the Memory Buffer* from either of two twelve-bit sets of gates.
5. *Clearing of A* conditional upon externally generated signals.
6. *Gating outputs* from the Accumulator to external equipment.
7. *Transferring information* from the Memory Buffer to the Accumulator under the control of external signals.
8. *Controlling high-rate inputs and outputs* between internal core memory and external equipment.
9. *Reading of keyboard.*
10. *Reading of console switches.*

Hewlett-Packard and Personal Computing Systems

Charles H. House
Hewlett-Packard

Abstract

Scientific computing systems for individuals were pioneered early at Hewlett-Packard, beginning with the 9100A Desktop Calculator in 1968. Extensions of this first machine were soon seen in Personal Peripherals, such as Printers, Tape Cartridges, and Plotters, and followed by Graphic CRT Displays. By early 1972, the Desktop unit had been augmented by a very powerful Pocket Calculator, the ground-breaking HP 35A.

This paper traces the evolution of these machines to the present day, following four main themes: 1. Scientific Desktop Computers; 2. Personal Calculators and Portable Computers; 3. Personal Peripherals; and 4. Personal Business Computers.

The underlying technologies are described, as well as the product definitions and the market development. The company organization is described in some detail, since it appears both unorthodox in the industry and particularly appropriate for certain elements of the accomplishments.

Some of the key people will be highlighted for their personal contributions, but the theme will revolve rather around the Team structure that seems responsible for the continued success in many of these product sectors.

The paper concludes with considerations about two corollary impacts of this pioneering effort: 1. The development of Data-Domain design and test methodologies; and 2. The impact of Pocket Calculators on the mathematical skills and procedures of our society.

A History of the PROMIS Technology: an Effective Human Interface

Jan R. Schultz
PROMIS Information Systems, Inc.

1. Introduction

It is perhaps unusual for a computer technology to grow out of a social movement, but that is what this story is about. A small group of medical and computer people worked together for 15 years developing a computer technology to support a medical philosophy. The grant proposal of 1967 which provided our original funding set forth the goals that were to guide us:

> We propose that: (1) the medical record, utilizing a "problem oriented" approach, be the instrument whereby the following objectives can be implemented, and that (2) the technique of record keeping described in previous publications (Weed, 1964, 1966) be the basis for the beginning realization of these goals and that (3) a real-time computer system be used to overcome the data distribution and time barriers that are insurmountable on a manual basis using conventional hospital and clinical medical records.
>
> Objectives: To develop a system that will:
>
> 1) Facilitate good patient care by making immediately available (in minutes) to the individual physician a complete, updated list of problems on any patient and by providing simultaneously, as a unit, all the data in sequence (narrative, laboratory, etc.) pertinent to any of these problems.
>
> 2) Make possible epidemiological studies and other research endeavors in terms of problems, having all the data on any given problem immediately available.
>
> 3) Make possible a medical audit whereby the standards of care being provided for a given entity (e.g. hypertension) can be rapidly assessed because of the specific orientation of all the data.
>
> 4) Make possible a business audit to assess the physical, financial and time resources that go into the solution and management of a given problem. The need for a more organized, efficient and economical approach to the management of common medical and surgical disorders may then be documented.

These remained the objectives of the project throughout its history.

I met Lawrence L. Weed, M.D., in November of 1966. Weed had published his initial paper on a new organization of the medical record: "Medical Records, Patient Care and Medical Education" (Weed 64). He was interested in exploring the implications of automating the new organization of the medical record. We decided to work together for six months and see if during that time a longer term plan of collaboration could be worked out.

When we began our work, other medical record research groups took the dictated or written words of physicians and other medical personnel and manually entered them into a computer. We decided from the very beginning to interface the information originator directly to the computer and develop techniques and tools to facilitate this direct interface. We decided on this direction for a number of reasons: among them first, the time lag between the dictation and the entry of the medical information could have a negative impact on the patient care delivered, and second, the use of the computer as a recall and structuring mechanism for medical knowledge and medical records could allow the medical personnel to

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0159 $00.75

operate from a universe that potentially could be much larger than what they could remember, and the data gathered in a specific area of the record could be consistent across all health practitioners entering it. The decision to directly interface the information originator brought us directly to the major areas of our research: the form of the human interface and the representation of medical knowledge within the computer.

Using computer technology to store and retrieve the medical record required the medical knowledge entered into the computer to have a rigorous structure. The medical community considered the notion of structuring a present illness, a medical history or a physical examination tampering with the art of medicine. Developing outlines of the structure of the various elements of the medical record was only the beginning of the effort. We had to find physicians, nurses, pharmacists, laboratory technicians, radiologists and other medical personnel who were not only organized in their thinking, but willing to expend enormous mental effort to fill in the outlines with medical knowledge to make the system usable. The medical group in the PRoblem Oriented Medical Information Systems (PROMIS) Laboratory was such a group of people.

The structure of the Problem Oriented Medical Record (POMR) served as our guide in defining the organization of all the data in the patient's medical record. The structure corresponds to four medical actions. The first is collection of a Data Base; from this the Problem List is formulated; for each active problem Initial Plans are written; and each problem is followed in Progress Notes. Without this structuring mechanism, our job would have been much more difficult because the traditional medical record has no logical structure but organizes data by its "source."

The logical POMR structure requires that the same data be retrievable in many different ways. It is necessary, within a patient's record, to look at all progress notes for a single problem or all progress notes for the last 24 hours or a tabular array of specific physiologic parameters (a flowsheet). The requirements of the POMR were beyond typical data base management systems of 20 years ago (and are beyond most of them today).

The medical record contains both narrative and numeric data. The narrative data of the medical record is variable length and the numeric data has many forms. It was impossible to define a fixed length container or set of containers that could encompass all the data in the medical record. It was necessary to manage variable length text strings that could be retrieved in different orderings to different output devices. The numeric data contained in the medical record had hundreds of forms and had to be retrieved as part of a narrative note as well as part of a flowsheet. Both narrative and numeric data needed encoded information associated with it, to facilitate the various retrievals and to make the data a legal medical record.

The initial decisions to interface the information originator directly to the computer system and to use the POMR to structure the record and provide an outline for the guidance within the computer system carried the medical and computer groups far into computer technology research and into advanced exploration of knowledge base development for the representation of medical knowledge.

2. A Search for the Correct Form for the Human-Computer Interface

2.1. Initial Tools for the Human-Computer Interface

The initial plan for the entry of medical record information was to use optical scan sheets and the IBM 1092-1093 programmed keyboards with an attached typewriter keyboard. Optical scan sheets were designed for the past medical history and systems review as well as for the physical examination. The programmed keyboards plus the attached typewriter keyboard were to be used for entry of all other information. The programmed keyboards accepted opaque plastic overlays or keymats placed over an array of 26 columns of 10 keys each, for a total of 260 labeled key locations. Each keymat could be removed and the programmed keyboard could sense the particular keymat being used, thereby identifying the type of entry into the system. Keymats were designed for each major medical subspeciality and were organized minimizing the changing of keymats and maximizing the amount of information that could be entered from the keymat without having to use the typewriter keyboard. All information entered using the programmed keyboard was printed on the attached printer so that it could be verified before being stored in the computer.

During the first six months of 1967, several thousand

past medical histories and systems reviews were processed, using two trained interviewers to fill out the optical scan sheets for entry into an IBM 1440 computer. The house staff at the Cleveland Metropolitan General Hospital were trained to fill out the physical examination scan sheets, and several hundred physical examinations were done. Great care had to be exercised in filling out the sheets or the optical scanner would reject them. The house staff had difficulty filling them out, and because of the large number of rejections, the use of the scan sheets was discontinued.

During the summer of 1967, Charles Burger, M.D., the designer of the keymats, tested the feasibility of using the programmed keyboards to enter a complete medical record. Burger's work was conclusive; the programmed keyboard presented the user with too much information, and the subsequent search time to find the proper key was excessive. The search time combined with mat changing time resulted in slow operation. A sample record took six hours to enter, at least double or triple the time required to write the original record. Scan sheets and programmed keyboards didn't solve the problem of effective entry of narrative data.

2.2. Expanded Search for Appropriate Technology

Because readily available off-the-shelf hardware was inadequate, we decided to investigate hardware and software approaches that could solve the difficult problem of an effective, facile human-computer interface whether or not the technology was commercially available or economically justifiable at that time.

During our expanded search for an appropriate technology, we met Robert Masters and Harlan Fretheim of the Research Division of Control Data Corporation. They jointly developed a cathode ray tube (crt) terminal with a touch-sensitive screen, the Digiscribe, with software to support it. The software consisted of three inter-related programs: the Human Interface Program (HIP) managed user touch-screen selections, the Selection Element Translator (SETRAN) managed the entry of new frames of information into the frame dictionary, and the SHORT operating system managed the execution of programs called from selections.

2.3. Initial Use of a Touch-Screen Terminal

From August of 1967 through February of 1968, we used a Digiscribe terminal connected over phone lines to a Control Data Corporation 160-A computer in Lansing, Michigan, at the St. Lawrence Hospital. Initial versions of the Human Interface Program and the Selection Element Translator were available, and the new software tools required that the programmed keyboard keymats be restructured. This was to be the first of many conversions of the medical content to fit it into an expanded and more capable technology designed to enhance its use. In April of 1968, we installed a Control Data Corporation 1700 computer at Cleveland Metropolitan General Hospital.

The Human Interface Program and the Digiscribe touch-screen terminal allowed medical personnel to be directly interfaced to the computer system. An array of choices was displayed on the screen and users selected one by touching their finger to the screen over the selection. For each of the twenty selection areas, the Digiscribe generated a different character which the Human Interface Program could interpret. Based upon information in the frame, a branch to another frame could take place and new information could be displayed to the user. (See Figure 1.) During normal selection processing, the keyboard was used for special functions such as erasing the last selection, aborting the current message and quickly exiting the system.

```
CHEST PAIN:  03463

-------------------------------------------------------
ONSET:  *                    COULDN'T DETERMINE.

GRADUAL (INSIDIOUS).         DIDN'T DETERMINE.

SUDDEN (ABRUPT).
```

Display in HIP

```
RI03463 03463AC M
ONSET:  *                    COULDN'T DETERMINE.
 00000 00000 X 000 0         -03463 00000 X 000 0
 F000002
GRADUAL (INSIDIOUS).         DIDN'T DETERMINE.
-03463 03434 X 000 0         -03463 00000 X 000 0

SUDDEN (ABRUPT).
-03463 03434 X 000 0          00000 00000 Q 000 0
```

Same display in SETRAN

Figure 1. Message generation frame. The "F" parameter associated with "Onset" defines the output format for the title "Onset."

Selection processing used an internal stack of frames waiting to be displayed as well as user written programs that extended the normal selection processing. The result of a user's selections was a message displayed on the top three lines of the screen as well as other information not seen by the user but used by programs to

interpret the selections. This unseen information included the frame number, the choice number within the frame and internal parameters included with the selection on the frame. The internal parameters were used to code selections so that programs interpreted compact codes rather than alpha-numeric data. The internal parameters were the coupling mechanism between the users making selections on the screen and the programs to store, manipulate and retrieve the medical data.

The Selection Element Translator was a program to edit frames. It was used to enter new frames and alter existing ones. The branching among the frames was also defined by the Selection Element Translator. The frames were organized into three types and contained selections that could fit on a single screen. A frame was a physical unit defined by the screen size. The frame types were defined by the function they were to perform, and it was not possible to mix these functions among other frame types. The three functions were: the starting of new messages (and the termination of old ones), the generation of a message and the display of textual information. The first type displayed an array of 7 x 2 choices, the second an array of 6 x 2 choices with the top three lines for the display of the generated message and the third consisted of between 1 and 19 text lines.

We trained physicians and other personnel to use the Selection Element Translator so that the medical knowledge could be entered into the computer without a computer person acting as an intermediary. This ability was fundamental during the development of the system because of the massive number of displays that had to be entered.

The SHORT operating system supported multiple terminals as well as application programs operating in a multi-programming environment. The frames, application programs and all medical data were mass-memory resident. Most selections made by a user required four disk accesses, and the station state data were resident in central memory only while the selections for that station were being processed. SHORT supported four classes of application programs: the first two allowed extensions to the Human Interface Program, one class allowed keyboard input (e.g., the Selection Element Translator), whereas the other did not; the next application program class ran in parallel with the user at the terminal and could communicate with the terminal (e.g., a retrieval program to display information to the crt); the final class executed in the background after the user signed off the terminal. The data generated by the user could be passed to each program class.

The hardware base consisted of a Control Data Corporation 1700 computer and associated peripherals. The CDC 1700 had 32k 16 bit words of central memory with 1.1 micro-second cycle time, and the cpu executed most instructions in 2.2 microseconds. The mass-memory was CDC 854 disk drives, each holding 7.2 million bytes of storage and each with an average seek time of 70 milliseconds. The Digiscribes were connected to the cpu through a display controller with a delay-line for each terminal. The maximum distance between the display controller and a terminal was 1000 feet. The transfer rate between the cpu and the display controller was 50,000 characters per second. The crt screen had 20 lines of 50 characters each. The Digiscribe consisted of transparent, conductive pads on the crt screen and the associated electronics sensed which of the 20 pads were touched. Each pad covered the left or right 25 characters of every other line.



Jan Schultz at a Digiscribe showing a visitor the system.

2.4. Initial medical content development

Developing a system that enabled a physician, nurse or other medical personnel to enter narrative medical information using a touch-screen as the primary interface was a major challenge. Stephen V. Cantrill, M.D.,an IBM Fellow during the summer of 1967 (before starting Case Western Reserve Medical School), analyzed a large number of manually recorded Cardio-Vascular problems and defined 90 frames which together could be used to enter most Cardio-Vascular problems. Also, during the same time period, the frames defining present illnesses for GI/Liver

problems were written.

Because of the limitations of using the optical scan sheets for patient questionnaires (which included no branching capabilities, interviewer administration and a high rejection rate), we investigated the feasibility of having the patients use the touch screen for eliciting historical information. Werner Slack, M.D., a medical researcher at the University of Wisconsin using a Linc computer, had demonstrated that directly interfacing the patient with a crt terminal for data collection was feasible for a selected patient population. We questioned whether we could do it routinely for all patients. We developed two short questionnaires for self-administration and found that except for people with poor eyesight, with poor reading skills or with limited ability to read English, the questionnaires could be used easily. Children down to the age of 10 years could also use the touch-screen terminals to answer the history questions.

2.5. Initial electronic record development

The programs to store and retrieve all types of medical data were evolving in parallel with the experimental entry of the narrative medical data using frames. All medical data that we thought could be effectively stored in an electronic record were classified as either narrative or numeric. Narrative data was further classified as resulting directly from selections or resulting from answering questions (translated data). There were three types of data stored in the medical record: direct narrative, translated narrative and numeric data. Keith G. Morgan, a Ph.D student in the Engineering Program at Case Western Reserve University, joined us at Cleveland Metro General Hospital. Morgan developed a program to store numeric data and wrote retrieval programs to display the numeric data in flowsheet form. Cantrill developed a program to translate a questionnaire's responses into narrative text for storage into the patient's record. He also developed a program similar to the Selection Element Translator for the entry and modification of dictionary elements used in the translation process. Lee Stein, a researcher from the original Bolt, Beranek and Newman "Hospital Computer Project" and part of our group in Cleveland, developed a program to transform the user's selections into narrative text for storage into the patient's record. I developed routines to assist in the display of the stored data. One routine, called FORMAT, interpreted codes stored with the narrative data so that the internal form of the record data was output device independent.

The ability of the Human Interface Program to couple narrative selections with internal parameters stored in the frame allowed the frame developer to enter the parameters needed to encode the data and to define the data's output format at the same time the content in the frame was entered.

2.6. Initial phase of development

The initial phase of the development focused on the correct form for the human interface. Other options were tried, but very early on we settled on a crt with a touch screen as the ideal interface for both medical personnel and the patient. At the time we did not fully realize that the interface was effective not only because of the touch-screen, but because of the high data rate of 50,000 characters per second between the cpu and the terminal and the system's ability to process most selections in under .5 seconds.

3. In Search of the Right Development Environment, We Move to Vermont

In July of 1969, a group of five families moved from Cleveland, Ohio, to Burlington, Vermont, and PROMIS Laboratory was formed. One of the original goals for the automation of the POMR was the integration of the patient's bill with the clinical medical record. The hospital in Cleveland was a county hospital, and while we were moving from the initial phase of our research to a phase requiring access to more of the hospital, the hospital's data processing department was shifted to the county data processing department. This meant that there would be no opportunity to integrate the electronic medical record with the patient's bill and the hospital's administrative operation. We began looking for an appropriate site to continue our research.

The University of Vermont's Medical School and Medical Center Hospital offered us the opportunity we were looking for: a teaching hospital closely associated with a University through which our funding could continue and an assurance that the data processing department at the hospital was progressive and willing to work with us.

4. The Medical Knowledge Data Base in the early 1970's

The medical data base consisted of two large bodies of knowledge: knowledge that traditionally resided in medical libraries, journals and textbooks, and knowledge concerning the individual patient. The generic medical knowledge was housed in frames; patient-specific knowledge resided in electronic patient records. This section will describe the frame based generic medical knowledge.

Getting the initial PROMIS system operational on a medical ward in 1970 required a massive amount of medical content to be available on the frames for selection by the medical users. Drs. Larry Weed and his wife Laura B. Weed worked together translating medical content into a form which could be displayed for a medical user. The first step was to create a skeleton structure, as a first approximation, which was a synthesis of remembered experience, a review of classical textbook descriptions of disease, and a review of the appropriate current literature. This process was completed in a year. Once the initial framework was complete, consultants in various subspecialities were used to help continue the building process.

4.1. Drug Information Frames

The drug information frames were constructed by George Nelson, MD., and Genevieve Gilroy, registered pharmacist. The drug information frames provided the medical user using the system with the basic facts about commonly used drugs. The guide for entry of the initial information was focused on the common drugs since most adverse reactions occurred with them. One hundred and thirty of the most commonly used drugs at the Medical Center Hospital of Vermont were compiled, and by 1972 over 180 separate drugs were represented in the system. The information was largely abstracted from standard textbooks and articles in pharmacology, with supplementation from the current literature. The drug data was organized in displays entitled "Check Problem List for," "Side Effects to Watch for," "Drug and Test Interactions," "Usual Dosage," "Mechanism of Action," and "Metabolism and Excretion." All information was reviewed by other physicians interested in clinical therapeutics.

4.2. Diagnostic Process

The initial system was used not only as an aid in making therapeutic decisions, but also in arriving at diagnostic plans. George Nelson did the initial work on the diagnostic plans. This was not diagnosis by computer but used the computer's recall ability and the electronic medical records to assist the physician in the diagnostic process. The first step in the process was the creation of problem formulation sequences for most medical problems and the association of each problem with a unique number, called the problem plan number. Plans were broken up into Get more information and Treatment. The Get More information section was further broken up into For More Data Base, Diagnostic Process, and For Management. The Diagnostic Process was further subdivided into three sections. A single problem plan number could have information about six or seven aspects of the plan.

By August of 1973 there were problem specific diagnostic and therapeutic plans for over 600 problems covering a wide range of medical problems. The plans were audited by outside medical authorities to assure the safety and appropriateness of the decision pathways. Most of the frames were documented by reference citations and the citations were available in the Medical School Library. We did not have enough mass-memory space available to keep the citations on-line. By this time the drug displays had been expanded to include over 300 individual agents.

4.3. Radiology Reporting

The initial system also included frame based medical knowledge for use in support of the radiology department. Peter Dietrich, MD., a radiologist, developed the medical content in this area. The x-ray plans for many problems formulated in the computer were built. A structured reporting format that was both problem and radiologic procedure specific was developed. Reporting sequences for over 150 problem/procedure pairs were built by October of 1972.

4.4. Problem Plan Numbers and Tables

In order to accommodate the problem specific information in the system, a new container for medical knowledge was necessary. The information in the container was never seen directly by the user, but was needed to collect together all the frames that related to one problem plan number. The new container was called the branching information table and it had slots for each type of problem specific information. Each slot contained a frame number that could be put into the branch stack that the Human Interface Program used, so that if a specific selection were made on a frame and the patient's problem were known, then the problem plan number could be used to look up the correct branching information table slot

for the selected problem. The branching information table was the first of many tables that would come to exist as part of the medical knowledge data base.

The table is the container for all machine-readable data related to a medically defined problem or procedure or other entity such as a system user or terminal. Each table has a unique address or code. Most of the data in a table is not visible to the medical user, but is used by programs to perform actions or do special display branching.

5. An Operational PROMIS on Medical and Gynecological Wards

The PROMIS system replaced paper records with a problem-oriented electronic computer record for six months on a general medical ward and nearly four years on a gynecology ward in the Medical Center Hospital of Vermont. This development took place from July of 1970 through November of 1974. More than 3000 patients were cared for using this system during the time it was in operation. More than 500 individuals (physicians, nurses, medical students, house staff, radiologists, pharmacists, social workers, ward secretaries, dieticians and laboratory technicians) used PROMIS as the sole mechanism to add and retrieve information in patient records. The prototype PROMIS system demonstrated solutions to many general problems seen in other computer applications. Through redundant hardware, careful maintenance, and good diagnostic tools, the system was made available to users more than 99.6 percent of the time, within a scheduled 24-hour, seven-day week operation. The system was designed for fast response to facilitate use, and would respond to a user's selection by presenting another screen within 600 milliseconds for over 50 percent of the selections. Response was both rapid and comprehensive as patients' complete records were available on the terminal during their hospital stay as well as more than 30,000 displays of medical information available for guidance.

Eligible personnel throughout the institution were allowed access to the electronic record. Terminals on the ward, in the operating suite, in the pharmacy, in the clinical laboratory and in the department of radiology demonstrated the synergism possible using an electronic record.

The overall goal during this time period was getting PROMIS operational on a single hospital ward and analyzing the operational hardware and software system in enough detail to allow realistic estimates for the size, cost and system structure necessary to implement PROMIS throughout a hospital.

5.1. The Software to support an operational PROMIS

The software developed to support the operational PROMIS included the SHORT operating system, the Human Interface Program and the Selection Element Translator, supplied by CDC. The Store program transformed the data generated by the Human Interface Program into patient record and other system files. The Retrieve program abstracted information from the patient's record or other system files and displayed the abstracted data in the form of a narrative report or a flowsheet. When all of the application programs were written for the CDC 1700 system, there were over 250,000 lines of assembly language code. (Schultz, Cantrill, Morgan 1971)

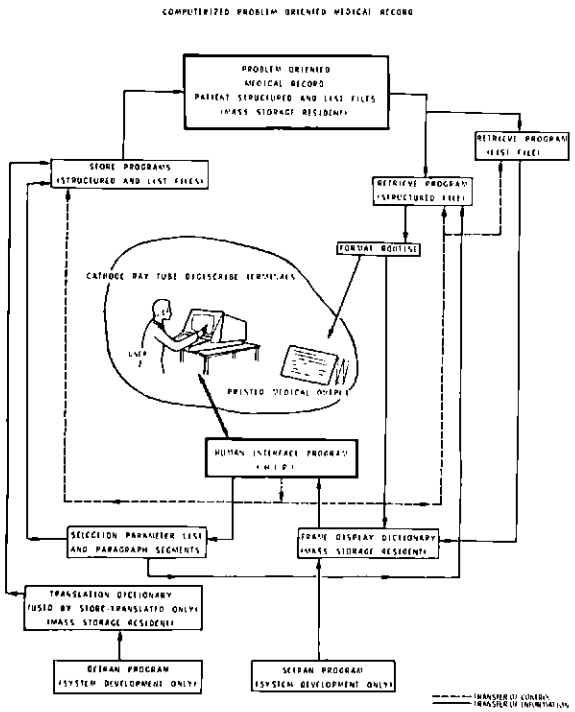


Figure 2. The Elements of the Application Software.

5.2. The Form of the Data Within the Electronic Record

An individual patient's electronic record was mass-memory resident and required a structure that facilitated its manipulation while minimizing the number of mass-memory accesses required to perform the manipulation. An individual record consisted of one fixed length index block and a variable number of fixed length data blocks. The index

block was an index to all the data in the data blocks and was accessible using the patient's system identification number. The index block contained a variable length list of pointers to the data blocks along with other information to identify the type of data in the data block. To access data in the patient's record, a search was performed through the index's data block pointers looking for those pointers that satisfied the retrieval request. For any single data item retrieval, two mass-memory accesses were required.

The internal form for a data item within the electronic record depended upon whether the item was narrative or numeric. For narrative data, the internal parameters associated with the selections were interpreted as format codes to be used to define the retrieval format. For numeric data, the internal parameters associated with the selections were interpreted as data types and each data type was stored in a separate fixed length field. Consider the following example: a temperature's internal parameters were an internal code number, a time (10:23), a date (February 23, 1970), a title (Temperature), a numeric value (38) and a numeric qualifier (degrees Centigrade).

5.3. The Selectable List Files

To facilitate access to rapidly changing patient record information, it was necessary to present the patient information on displays for selection. A list of patients on a ward or the list of current medications for a patient are examples. Displaying the lists for selection required that the data to be displayed be readily available so that it could be retrieved responsively. The selectable list files were an abstraction of data stored in the patient's record to facilitate the rapid retrieval of selectable lists.

5.4. The Medical Guidance To Support An Operational PROMIS When health care providers sat down to use the terminal, they first selected from a list of patients presented on their terminal screen the name of the patient with whose record they were to interact. Having chosen a patient, the providers selected one of the sections of the medical record - the Data Base, Problem Formulation, Plans and Progress Notes - in order to add to or retrieve from that part of the patient's record.

The Data Base

The nurse entered the patient's chief complaint, patient profile and the general appearance and vital sign components of the physical examination. The patients, after only two-to-four minutes of instruction, entered their own review-of-systems by answering a series of up to 275 yes/no and multiple choice questions. Only the positive responses were stored in the record. The Review-of-Systems was then retrieved and reviewed by medical students and house officers. Positive responses were developed into present illnesses by touching selections specific for each symptom. The physical examination would then be entered using up to 1410 frames, the number actually used being a variable related to the patient's physical status.

Problem Formulation

After completion of the Data Base, the physician formulated the patient's problems by making a series of selections on displays. Each problem would be stated at its level of definition, ranging from a symptom to a firm diagnosis. The displays facilitated further description of problems by guiding the user - frequently providing them with criteria to help use consistently precise language. These criteria or definitions were taken from the literature with a library reference citation noted on the display. The system facilitated easy updating of any problem when new information gave added insight. Problems could be added or their status changed (e.g. from active to inactive) at any time.

Plans

Each component of the plan was individualized for many problems in the system. The system contained displays based on the current literature detailing how to work-up, treat and follow the problem. These displays removed reliance on memory and allowed a new intern to work-up the problem with a degree of accuracy and completeness comparable to a subspecialist. The displays did not constrain the users; they were free to order whatever they pleased - but the displays served to guide them. More than 600 specific disease entities had some degree of problem-specific information in the system.

Progress Notes

All notes written about any problem after the Initial Plans for a problem are Progress Notes. These included physicians' notes, nurses' notes, and notes of social workers, chaplains, and all who had a recorded contribution to the patient's care. All Progress Notes were entered by first selecting the appropriate problem on the terminal

screen from the patient's list of problems. The Symptomatic and Objective sections took the user to frames specific for the body system of the problem. The nurses were encouraged to enter Progress Notes according to the severity and rate of change of each problem of each patient under their care. They used the same displays as the physicians, although some displays were used more by nurses than by physicians. Operative notes, radiology reports, clinical laboratory reports, social service consultations, and specialists consultations were also entered as Progress Notes under the problem they addressed.
In September of 1974 there were approximately 30,000 frames in the system. A breakdown showed that 6000 of these were Data Base frames; 4000 were Problem List frames; 7000 were Plans frames; 8000 were Progress Notes frames; Superstructure and others made up the remaining 5000 frames.

5.5. Data Retrieval

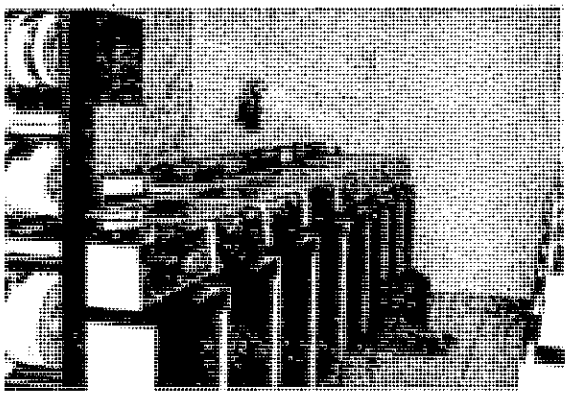
The above four sections comprise the basic patient record. This data was kept in electronic form in the computer system during a patient's hospitalization. There were no written records maintained in parallel. The updates to the record were printed each day and were available on the ward. Patient information could be retrieved at any time at any terminal by an appropriately identified medical user. Since the record was in electronic form, data could be retrieved in many ways. Information could be retrieved on a specific problem, on all problems chronologically, or in flowsheet format. Current outstanding drug orders, investigations and accumulated charges were also available by problem. The problem of absent or lost records was non-existent. The records were all legible. Any number of users could simultaneously retrieve from or add to a record.

5.6. Problem Oriented List of Charges

One of our initial goals, the creation of a Problem Oriented List of Charges was accomplished. We compared a hospital bill with a list of charges calculated from the chargeable items in the electronic record. We also presented a patient's problem list with the charges associated with each problem so that medical users and administrators could see the cost of treating each problem. We discovered that the Problem Oriented List of Charges created from the electronic record was different from the hospital's bill since it represented everything that was recorded in the record and not just those things that had a charge slip written for them.

5.7. The Hardware to Support an Operational PROMIS

The hardware base consisted of two Control Data Corporation 1700 computers and associated peripherals. Each central processing unit had 32k words of central memory and had seven disk drives with 7.2 million bytes capacity each. There were 22 local terminals and 8 remote terminals. Remote terminals operated at 4800 bits per second and contained their own delay line refresh memory. The remote terminal multiplexor communicated full duplex at 4800 bits per second. A modem at each end of a C2 conditioned, leased 4-wire phone line was used to connect the remote terminals to the terminal multiplexor. The 4800 bps modems cost over $10,000 each and had to be manually equalized to the telephone line.



This is 50 megabytes of mass-storage in 1970.



Ernie Preiss working on the CDC 1700.

5.8. Information Utility

The continuous operation of the system required the development of a reliable, backed-up hardware base that the users considered to be an information utility. One

of the major elements that contributed to keeping the system in continuous operation was our ability to see patterns in the occurrence of various system problems. Whenever a problem occurred that affected the operation of the system or more information on the problem was available, we described the problem in problem-oriented fashion and posted it so that periodically we could discuss the current operational problems. We kept paper logs of all operational problems since there was not enough mass-memory space available to keep the data on-line. Eventually, when more mass-memory was available, we kept the problem statements on-line. There were eleven major problem categories, divided into hardware, software, architectural relationships and undefined. Each category was divided into maintenance, operation, environment, financing, personnel, development and education. Problems in the hardware, central processor and peripherals operation category ranged from "core parity errors" to "idle time low for unknown reasons." Problems in the hardware, environment category included "Ernie's tools disappearing" and "cpu room floor dirty and place a mess." The total log for this time period included over 500 active problems. One of our mottos for the period was to "let the problem lead us," and we had enough problems; the only question was where it was to lead us and which problem would be the leader.

6. The National Library of Displays

The medical group within PROMIS Laboratory was formed to create and maintain the thousands of computer displays. The medical content was viewed as a textbook or encyclopedia organized and available according to the patient's problems. As Dr. Weed wrote in 1972:

> "The opportunity now exists in modern medicine for making available to health care personnel tools which will permit them to perform with excellence without depending on an encyclopedic memory.... It is therefore an appropriate goal to seek to make available to all health care personnel a tool which amplifies to a uniformly excellent level their initiative, memory, and ability to create and to execute reasoned and disciplined plans.... The goal can be accomplished in large part if the tool has built into it the parameters of guidance and currency of information required by health care personnel as they perform and record their work. Then, individual human memory and initiative will cease to be the critical link between all we know and all that must be done."

6.1. The Process of Medical Content Development

The process of creating the medical content involved much more than just entering the medical knowledge onto the frames. Prior to the release of the medical content for on-line use, a rigorous in-house audit was completed. Outside experts in the relevant medical subspecialties were invited to Burlington for a 3-5 day period, during which time they audited each frame sequence for safety, accuracy, currency and completeness. The audit comments were then reviewed by the author of the frame sequence and appropriate changes were made. Each step of the process was documented. Finally a check of the branching was done to verify that the sequence worked as planned.

6.2. Preliminary Study by National Bureau of Standards

The process outlined above was limited by the resources of the medical group of PROMIS Laboratory. It became clear that prior to the widespread distribution of systems with the characteristics of PROMIS, a national body should be created to assume the responsibility for the review and distribution process. This would insure that the systems represented accepted standards in medical care as well as computer performance. The National Bureau of Standards in conjunction with the Health Services and Mental Health Administration sponsored a study directed at identifying actions appropriate for facilitating the wide spread use and maintenance of PROMIS. Melvin Conway, PhD., was hired by the National Bureau of Standards to do a preliminary study.

The report from the study was available in October of 1972 and was an initial planning document. It raised many issues: display library accreditation, standardization, distribution, quality control, the administrative tools required by the library, certification in the field of the record-maintenance systems which are using the displays, and controlled evolution of the entire process.

The report defined terminology and concepts which we incorporated into our thinking and ultimately into our evolving system. It defined Generic system specifications which set forth criteria for initial certification for record maintenance systems so that system vendors could design to the specification. It defined under this a system line which is a definition of a class of concrete record maintenance systems and under this a system instance which is an individual medical record maintenance system.

A parallel three level hierarchy of definitions existed for the collection of displays. The generic library is the primary standard resident in the national repository from which all specific display collections residing in system instances are drawn. A sub-library is a subset of the generic library to allow distribution of non-identical display libraries. A library instance is a machine-readable data set of a sub-library intended to reside in one system instance. And finally, a library instance update is a portion of a library instance which is sent out to a system instance when only part of its display collection needs to be changed.

6.3. Release of Frame and Table Library

PROMIS laboratory was committed to seeing a National Library of Displays formed. We resisted efforts by the government and private industry to get early releases of the display library; we feared if multiple copies were released, it would become impossible to provide updates since the mechanisms for creating sublibraries and library instances had yet to be worked out. The merging of already released library instances with a library instance update was a difficult problem. The tools to perform the library instance update were not in place until 1979.

Our National Center for Health Services Research contract in 1976 required that a data description language be developed to externalize the frame and table library in a form that was machine independent. We made available on magnetic tape yearly the complete library as well as a copy of all of the computer software from 1976 until the last contract expired in September of 1981.

7. Expandable and Exportable Architecture Is Defined

Besides the continued operation of PROMIS in the hospital and the entry of new medical knowledge, a major effort was undertaken starting in the spring of 1973 to define an architecture that could serve a full hospital as well as be exported to other medical settings and different sized institutions. Wesley Clark and Mel Conway were hired by the National Center for Health Services Research to assist us in this task. The operational CDC 1700 served as a resource for defining the subsystems for a new architecture. The CDC 1700 system consisted of three subsystems: The Human Interface Manager had both hardware and software elements. The station hardware included the cathode ray tube terminal, the Digiscribe unit to make the screen touch sensitive, the hardware logic to make the terminal behave in the expected way and the display controller or multiplexor that interfaced the terminal and the cpu. The software to control the human interface interpreted the frames, concatenated the selections together to generate English narrative, determined the next frame and presented it to the user for selection and then re-initiated the cycle. The Patient Record Manager included the mass-memory to store the patient's record as well as the software to update, retrieve and take patient data on and off-line. The Frame Manager included the mass-memory to store the frames and tables, and the software to create and modify them.

7.1. Subsystems Characteristics

Each of the subsystems had characteristics which were used to helped us determine the means of instrumentation. These characteristics were determined by the way the system was implemented and by the way it was used within the hospital. The major characteristics of each subsystem are described below.

The Human Interface Manager

This subsystem had to be extremely responsive. We defined extremely responsive to mean that the time from user selection to complete display of the next screen's data was less than .25 second, 70% of the time. This responsiveness was necessary to facilitate the effective interface of professional non-computer trained people to a functionally complex computer system. The human engineering of the touch-screen is essential for the interface to work well. The data rate between the frame manager, the human interface manager and the station must be high. Only 50 milliseconds was allocated for the transfer time between the frame manager and the station. The displays used to generate data were sparse, with an average of six choices used out of a maximum of 14. The combination of the highly responsive system with sparse displays facilitates pattern recognition and rapid interaction by the user. Unlike other computer systems where the function of the human interface was to get to a computation, in our case the interface was the computation.

The Patient Record Manager

This subsystem must be extremely reliable so that patient data is never lost and so the many types of retrieval requests for individual patient records as well as for groups of records are handled responsively. Individual

records ranged in size from 10,000 to 100,000 characters (or more) with an average of 6,000 characters of data added daily. The access time to one element of a patient's record was 65 milliseconds, and it took from 2 to 200 accesses for most retrievals. An average storage into a single patient record required 6 accesses to the record file and 20 to 100 accesses to the input data generated by the Human Interface Program.

The Frame Manager

This subsystem contained 30,000 frames and potentially could contain two to three times that number. Over 18 million characters of storage were required, and the average number of characters per frame was 600. The 600 characters included not only data to be displayed to the user, but other branching and internal parameters. The internal form was not densely packaged; groups of fixed length fields were present whether filled or not. Two mass memory accesses were required to retrieve one frame. Twenty percent of all accesses were to two percent of the frames, so that a faster access time storage media for a small number of frames could have an impact on the through-put of this subsystem.

7.2. Potential Expansion of the CDC 1700 System

The CDC 1700 system could not be expanded to operate the total hospital, not only because it was technologically disadvantaged, but because the system software had many limitations that could only be changed with a major software restructuring. It is valuable to consider the types of walls present in that system. They became the guide to a redevelopment effort. Each wall was a design decision in the Human Interface Program or the patient record structure. The Human Interface Program's design restricted the maximum number of selections that could be made within one generated unit so that frame sequences had to be changed to keep the generated unit from overflowing; the frame library could contain a maximum of only 32,000 frames, and we were very close to overflowing it. The structure of the patient record restricted the size of the patient record index to a single mass-memory block so that some very long patient stays would overflow the block requiring the starting of a new record; the numeric information for one data item in the record had a fixed maximum size; a maximum of 144 patient records could be on-line at one time; the patient's problem and order lists had a maximum size so they had to be monitored and cleaned up as some of them reached maximum size; the numeric and narrative data were stored in separate parts of the record and retrieval functions that operated on narrative data couldn't be applied to numeric data and conversely. This made it impossible to retrieve a flowsheet of narrative data.

7.3. Incorporate Expanded Functions

It was also necessary to incorporate expanded functions into the system's operation. We wanted to be able to fully exploit the electronic record: by allowing its full access in all the supporting areas of the hospital; by performing data compression of the electronic record; by making available at all times the patient's problem list and current medications for all patients in our population; by developing a patient record that could span multiple admissions and serve for the patient's lifetime; and finally, by linking the medical data in the record with the patient's financial data.

7.4. Scale to Support Total Hospital

The new system had to be able to support the total hospital. The size of the system was determined by the number of active patients in the system at one time and the total served by the hospital, by the size of an individual patient's record, and by the number of terminals active at one time.

The Medical Center Hospital of Vermont in 1973 handled 20,000 admissions and 120,000 out-patient visits per year. There were between 25 and 85 admissions per day and 10,000 out-patient visits per month.

The size of a single patient admission was determined by looking at a "typical" six-day period of system operation and extrapolating. We used figures for the number of admissions per year and out-patient visits per year to arrive at the amount of patient record growth expected in one year. It was 1.24 billion characters or the equivalent of 4 double density IBM 3330s (the measuring stick of that era).

The number of terminals for the hospital was calculated based upon 4 terminals per 20 bed ward and 7 terminals per 34 bed ward and terminals in all the supporting areas. The total number was 200.

7.5. Architectural Characteristics are Defined

Given the characteristics of the functional structure of

the CDC 1700 system and the characteristics of the site for installation, an architecture for a PROMIS system was defined. The major elements of the PROMIS architecture were redundancy of hardware elements for reliable service, guaranteed responsiveness with minimal sensitivity to load, and access to any patient's record from any terminal within the system. The architecture had to support the locational diversity of the health care system since health care is not practiced in one geographical site, yet it was important to allow communication among the sites.

7.6. Two Remaining Issues

Besides the elements for the architecture listed above, there were two issues that had to be explored before a final decision on an architecture could be made. The issues were: should the subsystems be functionally partitioned into separate hardware elements, and should the patient record files be centralized or distributed?

One potential architecture was a network of mini-computers. Each node of the network was to be one mini-computer that would handle between 10 and 30 terminals. The number of terminals per node would be determined by requiring a system response of less than .25 second 70 percent of the time. The nodes would each contain the three subsystems. A communications medium would allow the nodes in the network to communicate. The patient records which were normally accessed by the terminals connected to one node would be contained in that node. Other nodes' access to a patient record would be on-demand and would require a transfer across the network. For storage into a record, packets of data would be shipped to the node which contained the patient's record.

Another proposed architecture was a functional partitioning of each subsystem into separate hardware elements. There would be Frame Engines containing the frames in a read-only form. Each Frame Engine would be a small mini-computer with mass-memory to contain the frame library and communication ports so that the Human Interface subsystems could communicate with it. The Human Interface subsystems would contain no mass-memory and would multiplex many terminals. Each Human Interface subsystem would have communication ports for connections to both a Frame Engine as well as a Record Processor. The Record Processor would be fully duplexed mini-computers with communication ports for connection to the Human Interface subsystems and to remotely located printers.

The issue of whether the patient record file should be centralized or distributed was posed in the following way: can we build a system that can serve New York City as easily as it serves the City of Burlington and the State of Vermont? Ultimately there would have to be a medical record network serving a very large region; the question was, shall we learn the problems of building such a network while it was small or wait and solve the problems posed by such a network only when it was no longer feasible to keep the data files centralized?

A paper simulation was done of the system data traffic among the subsystems, assuming the architectural model of a network of six mini-computers with each mini-computer managing 50 terminals and with intra-node traffic for a distributed patient record file being 25% of all accesses. The average total data rate was 42,000 characters per second; assuming a burst rate of ten times that rate and assuming maximum intra-node traffic for the patient record files, the data flow would require only 25% of a million word per second I/O bus. The remaining bus capacity would be available for instruction execution, data pool swapping and program loading. Given the system data traffic requirements and the type of hardware available at the time, there was no need to segment the functions into separate hardware units.

7.7. Final Decision

Our decision was to define an architecture consisting of a network of mini-computer nodes with each node containing the three subsystems. The patient record files would be distributed across the network with the records for the patients served by the terminals connected to the node defining where in the network the patient records would be located.

The architecture supported a distributed patient record file since ultimately not all records could be kept in one centralized system and we wanted experience handling a distributed patient record file while the files were of reasonable size. We wanted to solve this problem by design, not by being forced to consider it when the centralized system could no longer expand.

8. Specify Technology to Support the New Architecture

With the architecture defined by the end of 1973, our next year's jobs were set out before us. A mini-computer, terminal, inter- and intra-node communications bus as well as a programming language had to be specified and selected for the new PROMIS system. We had been looking for a hardware person to round out the PROMIS computer group when James Wanner walked through the door. An astronomer and mechanical engineer with digital electronics experience, his first task was the writing of a terminal specification. We decided to use MOS technology where it could be applied. MOS was "mostly off the shelf," and those items we could not find on the shelf we tried to put on the shelf. Detailed specifications were written for all of the items and we set things in motion to begin the procurement process.

8.1. Mini-computer specifications

The mini-computer had the following specifications: central memory of at least 128k, 16 bit or greater words with a cycle time of one microsecond or less, and direct memory access to peripherals; the environment protection had to include power failure restart circuitry and error checking on all data paths in the system; multi-level interrupts; memory protection consistent with any memory mapping or multi-programming executive; mean time between failure of less than one processor failure per year. Mass-memory was divided into 10 millisecond fast access for 2 million characters and 65 millisecond access for 200 million characters. Communications subsystem (four wire or phone) must support printers and inter-computer communications at 50,000 bits per second or faster. Up to fifty local touch screen terminals operating at 16,000 characters per second or faster had to be able to be connected to the system. Other standard peripherals had to be available.

8.2. The Touch-Screen Specifications

The touch-screen terminal had the following specifications: 8 1/2 inches high by 11 inches wide screen; two display modes - the first - 24 lines by 80 characters each, with 12 lines aligned under the touch-pads on the screen, and the second - 36 lines by 100 characters each with the bottom line aligned with the bottom line of touch-pads on the screen; alphabetic characters shall appear in at least three distinct ways; if a cathode ray tube is used, the display shall be flicker-free for all users; the touch pad screen shall have an array of transparent touch pads each covering a fixed block of characters in a displayed page of text. The pads shall respond to the operator's finger touch, independent of body position. When a pad is touched, the text lying under the pad shall be emphasized. This emphasis will continue until a new screen-full of text is received by the terminal for display. The bottom line of screen display text shall have 10 touch pads, each pad covering 7 characters, with a one-character-wide space between the pads. Eleven additional lines of text, on approximately 5/8" centers, shall have four touch-pads each covering 19 characters of text. Parallax errors between text and touch-pad positions shall be minimized by mounting the pads as close as possible to the text. Data flow into each terminal shall be at the rate of 15,000 characters per second, or faster. Data flow from each terminal shall proceed at a rate of at least 15,000 characters per second, or less if appropriate to the terminal controller subsystem. The terminal interface subsystem shall support as least 24 terminals in apparent concurrent active use. Each terminal shall have access to more than one central processor unit, for use as back-up on cpu failure. Portable terminals shall be provided by means of wheeled cart mounted terminals with single plug interface to the system and taps at various locations.

8.3. The Communications Subsystem Specifications

The communications subsystem had to be able to support up to 50 touch screen terminals operating at 15,000 characters per second full duplex, up to 20 printers operating at 9600 characters per second and at least one inter-computer connection operating at 50,000 bits per second.

8.4. Programming Language Specification

The programming language specification included the following description. A high level language is desirable, but not at the expense of execution speed or mass-memory access efficiency. The language includes character manipulation instructions. It is machine independent so that the application programs do not have to be rewritten when the machine is changed and so that the same programs can run on multiple machines. The language must be transparent to the operating system. The language must incorporate the ideas of structured programming. There can be separate languages for application programming and systems programming. Considering the characteristics of the original assembly language programs: very efficient, very difficult to change, very little documentation as part of the program; do better this time around.

9. The Architectural Elements

9.1. The Mini-computer Procurement

Request for proposals was sent to eight computer manufacturers. Four "no bid" the proposal. IBM said they had a "hole" in their product line and that a network of their computers would be too expensive. Digital recommended an 11/45, which met the hardware requirements, but would not make information available on their software. Although they offered to sell us hardware, they made no bid on our request for proposal. PRIME and INTERDATA made no bid.

We received four proposals. The Modcomp, Inc., proposal met the hardware specifications, but the system they proposed, the ModComp IV, was not yet available for testing. Control Data Corporation sent an extensive proposal; William Norris, the CDC President, delivered it and spent the day at PROMIS Laboratory discussing the continuation of our long term relationship. CDC proposed the most interesting system. It was a system composed of 10 peripheral processors from a CDC 7600 with a large shared central memory. However, the system was not a standard product, and a simulation showed that for our type of architecture the multi-processor system got in the way of responsiveness. The Data General proposal met the hardware specifications, but the software could only address 96k of the 128k address space so it did not fit the software specifications. Edson DeCastro, President of Data General, visited PROMIS Laboratory to discuss their proposal.

Varian Data Machines submitted the winning proposal with their V77-400 computer system. It fit both the hardware and software specifications and the company was interested in working with us on the joint development of a communications subsystem to support the high-speed touch screen terminals.

9.2. The Touch-screen Terminal Procurement

There were three departures from the main stream in our terminal specification: the touch-screen interface; the dual display modes: 24 lines by 80 characters for selection and 36 lines by 100 characters for text retrieval; and the high speed data transfer. After discussions with over 30 potential suppliers, we decided to drop the dual display modes as an essential requirement. We received eight responses to our specifications, and MEGADATA Corporation was selected to build the new high-speed, touch-screen terminals. The terminal was a random-logic PDP 8 so that it could be programmed to perform our specialized functions. It had expansion slots for the two special boards to be built by MEGADATA, one for the touch screen and the other for the high-speed line interface. The touch screen did not have set pad locations on the screen, but was an x - y digitalizer that had a resolution of .25 inch. It sensed a finger touch via echoing of surface waves. The first group of 20 terminals cost $8,500 each.

9.3. Communication System Procurement

The terminal-to-computer communications remained largely undefined until a computer system was selected in early March, 1974. Soon thereafter a suggested communications system was formulated, using the standard Varian Data Communications Multiplexor and VTAM software as a starting point. A research contract was signed with MITRE Corporation in July, 1974, to assist us in designing and implementing a communications system which would support digital traffic on a CATV-type distribution system for the PROMIS architecture. The resultant system was to provide highly reliable, low error rate and flexible reconfiguration capabilities for the PROMIS digital signals among the many terminals and several computer nodes. The data rate was to be at least 15,000 characters per second. Of secondary importance was the ability of the system to simultaneously support the communication of other services such as entertainment and educational television. MITRE was selected for this work because of their pioneering efforts with the MITRIX system which used CATV technology to build a time-division multiplexing scheme on a CATV bus for inter-node data transfer. They had developed CATV modems as part of the MITRIX system.

9.4. Programming Language Development

The only high level language available on the V77-400 was FORTRAN. Its code was not re-entrant, and it had an extensive run-time environment that was not appropriate for our tasks. We had done extensive programming language research. We wanted a language with syntax that would support structured programming, a semantics that would support the manipulation of logical records of variable type and length and a pragmatics that would support frames, records and indexes to both. We wanted to be able to manipulate strings that could grow to be very long (up to 32k characters) and to be able to access mass-memory resident data logically. Facile control of the touch-screen terminal was also a requirement, as was

network access to the logical data structure and network access to multiple communicating processes.

I designed a programming language based upon our CDC 1700 assembly language experience and a study of other languages. Morgan and I wrote an interpreted version of the language on the CDC 1700 with semantics as defined above. The interpreted version accessed the data using a structure similar to a list processing languages' "property lists." Extensive searching through the data structures was required for any operand access, and consequently it executed very slowly. It would "run like the wind" and could take more than 30 seconds to process one selection.

We decided to develop a compiler language, the PROMIS Programming Language (PPL). PPL was a combination of a high-level procedural language with a very powerful embedded database management system. It included procedures to manipulate the touch screen, schedule and sprout processes throughout a network, transfer data across the network, and manipulate strings effectively. It was efficient in terms of CPU cycles because it was not an interpreter; all data access to 16 bit numerics was in-line code and the internal form of the data required no searching. Each data element was accessed either with an absolute address or with a pointer variable and a relative address which was bound at run-time. Cantrill wrote the code generator, and I wrote the scanner and parser for the compiler. It compiled source code at 2,000 - 6,000 lines per minute.

The syntax of PPL was a major departure from "ALGOL-like" languages of that era. PPL's assignment operation was from left to right, the way one reads English. PPL had no explicit statement delimiters, that is, no ";" after each statement. PPL's comments could appear after implicit statement delimiters, and except for an * at the start of a new line, there were no explicit comment delimiters. Control statements all had explicit statement terminators. For example, an IF statement was terminated by an ENDIF or FI. Lists of statements could replace a single statement without requiring a BEGIN and END for the block. The syntax of a control statement required it to be spread across multiple lines, and no more than one statement could be put on a single source line.
The PPL syntax supported our ideas of what "structured" code should look like.

PPL was designed for the programming of applications. We could not afford to wait until the compiler was done before we began programming its run-time support routines. We were also concerned about the efficiency of PPL for systems programming tasks; since PPL code was to be machine independent, we were also concerned about distorting PPL to fit various systems programming tasks. For these reasons, Cantrill developed a structured preprocessor to the Varian supplied macro-assembler. Called STRAP, it allowed all assembly language code to pass through. Statements recognized as STRAP commands caused output of the appropriate macro-assembler instructions. STRAP was originally written in STAGE2, a machine independent macro-processor, but it processed code too slowly. STRAP was then written in itself and could process up to 5000 lines of code per minute.

10. Software and Hardware Specification

Our funding agency was devoting a large percentage of their budget to our development efforts and wanted to be assured that we were up to the task. An advisory committee was put together to oversee our development. The committee included Ivan Southerland, Allen Newell, George Robertson, Herbert Sherman and Jack Hall, M.D. Newell and Robertson had previously developed a menu selection system called ZOG, which sensitized them to our work. They applied the PROMIS interface ideas of a rapid response, large network system to ZOG.

The implementation of the new architecture involved coordinating efforts of multiple hardware vendors and the PROMIS Laboratory computer group. We had a limited budget and limited time. We followed D. L. Parnas's advice. The implementation used "specifications sufficiently precise and complete that other pieces of software can be written to interact with the piece specified without additional information... . The specification must be sufficiently formal that it can conceivably be machine tested for consistency [and] completeness [in the sense of defining the outcome of all possible uses]... By this requirement we intend to rule out all natural language specification." (Parnas 1972) We had been using decision tables for program development for a few years. Henry Beitz, a CDC consultant, convinced us that an unlimited entry decision table could be used as the basis for demonstrating an understanding of a problem and also as the fundamental design document. Decision tables were used to design and test the interface among hardware and software modules.

The run-time environment developed to support PPL (described below) was defined in 17 decision tables and 78 rules. The definition of the polled multidrop protocol to connect the high-speed terminals to the CPU was a set of 12 decision tables with a total of 77 rules. The protocol was used by Varian Data Machines, by MEGADATA, and by PROMIS Laboratory. When the hardware was delivered and the terminals were connected, the system operated correctly the first time it was tried. One problem was discovered in an error pathway in the hardware, and six problems requiring software modification were discovered. We also used "thin-wire" protocols to define our run-time environment. We sent messages among three separate tasks: the PPL code environment to manage the resources of the cpu, Terminal Input/Output to manage the terminal and printers, and Block Input/Output to manage mass-memory.

11. Run-time environment supporting PPL

The PPL run-time environment consisted of facilities to support the work requirement modularized into processes to, support the internal form of the PPL logical data type - the paragraph, and to minimize the number of mass-memory accesses.

11.1. Process Support

A process required access to three types of resources: the cpu for execution of PPL code, the mass-memory devices for access to the PPL paragraphs, and the terminals and printers. Each of these three types of resources was accessed using a task in the Varian Data Machines VORTEX II operating system. Based upon the resource required by the PPL code being executed, messages dispatched the PPL process among the different tasks.

11.2. Support for the PPL logical data type

Files and individual blocks were manipulated by the file system. The PPL logical data type was broken up into paragraphs, sentences and sentence elements and were manipulated by Sentence Input/Output routines. The file system, developed by Henry Stambler, used the VORTEX II software as a foundation and provided the following enhancements:

1) A file was configured as a set of discontiguous components spanning several VORTEX file areas or disk packs. Up to 64 components were allowed for each file.

2) New components could be added as needed.

3) A file could be as large as 2048K blocks. In VORTEX the maximum was 32K sectors. A block could be up to 2K bytes long.

4) The table defining the files resided in central memory so that any block could be obtained in a single disk access.

5) Software checksumming and read-after-write validation were provided.

The Sentence I/O routines, developed by Morgan, performed the mapping between the PPL operands and words within the file system blocks by maintaining state variables and a map-in area for each environment, making possible all the logical and string operations in PPL.

11.3. Mass Memory Access

PPL programs required rapid access to large amounts of data. It was impossible to keep even a small fraction of this data in central memory, and it was undesirable to access mass memory every time a piece of data was needed. A memory buffer pool, developed by James LeMay, housed all data accessed via PPL instructions. The data in the pool ages, and because a PPL program normally accessed the same information many times during its execution, the pool frequently still contained the information. The pool was a resource used by the resource management tasks; it was transparent to the PPL programmer. The ratio of block to disk accesses was typically between 2:1 and 25:1, based upon the type of program and the amount of data it needed.

12. The Library System: Management of the Medical Knowledge

Medical knowledge entry was controlled in a manner similar to patient record data entry and retrieval. The library builder makes selections at a touch-screen terminal. By answering questions and selecting the next step in a series of actions, the builder was guided through the steps required to add new information to the data base or update previously entered knowledge. The library system was developed to manage the frame and table library, in the same way that PROMIS was developed to manage the patient care functions. PPL was used to implement the library system on the same hardware base that the PROMIS system was implemented.

The library system managed frames and tables.

12.1 Tables

Tables were developed for many uses. The information in

a procedure table is used, for example, to:

1) determine branching to a subroutine of frames at certain points in the frame sequence;
2) define information that is specific to a site (such as drug prices and inventory levels);
3) define lists of logical actions that should take place when this procedure is stored in a patient's record;
4) specify lists of synonyms for the procedure to make it accessible on various alphabetic lists;
5) classify the procedure for all uses;
6) point to other data structures that are related to this procedure;
7) list all frames and tables that access this table.

12.2. Frames

The frames are interrelated in a network; each choice on each frame points to a frame to be displayed when that choice is selected by a user. The network is a guidance system in which every frame is simultaneously medical content and structure for the next finer level of content. The most general frames, superstructure frames, serve as high-level indices to other frames in the network, establishing the context (such as the current section of the medical record) the user will be working in. As the user progresses through the network, the frames become more specific, containing, for example, drug information or lists of diagnostic procedures, and enabling procedure ordering and reporting. (See Figure 3 for a frame sequence)

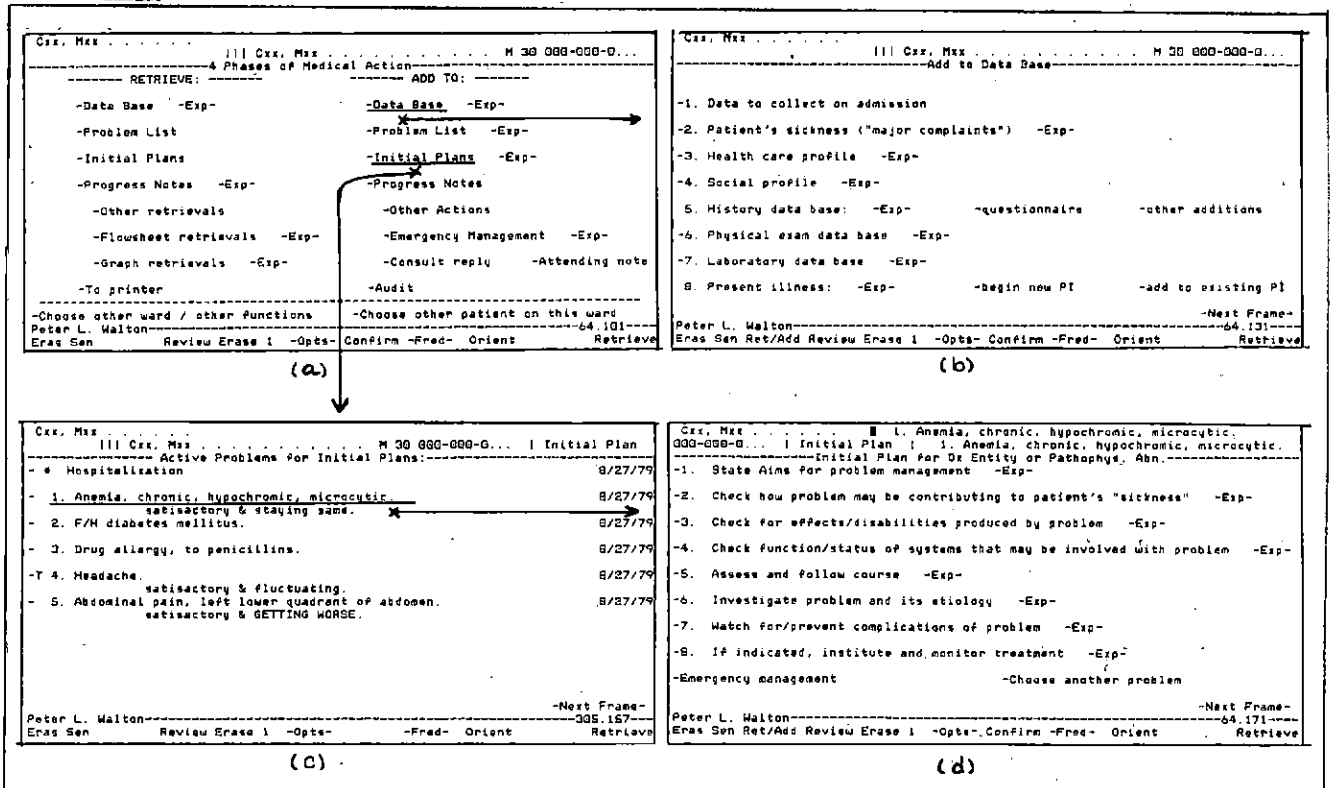


Figure 3. Example of guidance frames.

Frame (a) is the initial frame that was presented after a patient was selected from the list of patients whose records the user was allowed to access. From this frame, all additions to and retrievals from a single patient record can be performed. (Note the titles RETRIEVE and ADD TO near the top of the frame.) Selecting Data Base branches to the add to Data Base frame (b). Selecting Initial Plans, on the add side of (a), brings up a list of the patient's active problems. All patient care actions are taken in the context of a specific problem. If Anemia were selected, as in the example, the appropriate planning structure frame (d) is presented.

The displays contain frame content as well as context information for the user. The context changes as the user makes selections. The user's name is displayed in the lower left corner. Once a patient has been selected, the name is shown in the upper left hand corner. The selected problem is on the top line in the center.

12.3 Size and Maintenance of the Frame and Table Library

By 1979 more than 10,000 tables and 37,000 frames comprised the library. These frames and tables contained information on about 2700 medically defined problems, 640 drugs, 800 clinical laboratory procedures, 200 radiologic procedures, several hundred diets, activities, physical therapies and ward procedures, a comprehensive physical examination, patient history questionnaire, and problem-oriented superstructure.

Content was built in frames and tables from accepted texts and from articles in medical journals that satisfied selection criteria. More than 60 journals were regularly reviewed by medical librarians. Source, date, and builder of each piece of content was recorded in a reference. Auditing of existing content was performed continuously.

The frame and table library was maintained via an interactive frame and table editor (FRED - originally for FRame EDiting only), a syntax checker, a pathfinder, and other tools. The syntax checker reports violations of the rules used to build the frames and tables. The pathfinder finds all frames and tables that access a given frame or table. FRED manipulates the content in frames and tables. FRED is table driven: the type and form of each data element manipulated by FRED are specified in a set of tables accessed by FRED. FRED initiates backpointing and indexing among the frames and tables.

The frames and tables reference each other in order to define display branching or to use an entity. It is necessary to know how and where each paragraph is referenced to effectively manage the frame and table library. All backpointing is performed by a routine scheduled by FRED.

12.4. Capturing Medical Knowledge

The medical content form and structure evolved from emphasis on providing guidance to capturing the information needed to build the guidance capability. From 1976 to 1979 Drs. Peter Walton, Robert Holland and Stuart Graves worked with the assistance of Larry Wolf, a software specialist, on means of capturing and codifying the medical knowledge. What evolved was the idea of capturing the basic units of medical knowledge, the facts (Walton, Holland, and Wolf 1979). A fact can simply state a relationship among entities or qualify a relationship in greater detail. As was stated:

> "By stating relationships, facts provide a means for structuring medical knowledge. The structure can be viewed as a network of relationships. In qualifying relationships, facts capture the content of medical knowledge.
>
> "Facts are the fundamental data base of medical knowledge and are structured in ways that facilitate access to the data base. For example, the fact in Figure 4 is a "cause fact." Cause facts state an etiologic relationship between a predecessor (iron deficiency) and a successor (transferrin saturation) when the predecessor is considered sufficient to explain the successor (i.e. iron deficiency "causes" a transferrin saturation of less than 16 percent). ... Predecessors and successors are used to define links which the system traverses to access the knowledge pertaining to a particular entity.

```
Fact 135.00303
     Dz: iron deficiency
     Mf: transferrin saturation
     Value: below 16 % (average 7%)
     P[Mf|Dz] = 1

     Fact audited
     Pr (coded): iron deficiency
     Su (coded): transferrin saturation

     Flags: p[Su|Pr] = 1.

     Refs
       Clinical Hematology
       Wintrobe MM, et al., Eds
       Philadelphia: Lea and Febiger, 7th ed., 1974
         Ch 16: Anemias characterized by deficient hemoglobin synthesis and
             impaired iron metabolism
         pp 621-634
     Process: 134.1: Dependence fact from authority source
     By: RR Holland MD-SP: 1/17/79
     Documentation audited

     Refs
       Clinical Hematology
       Wintrobe MM, et al., Eds
       Philadelphia: Lea and Febiger, 7th ed., 1974
         Ch 17: Iron deficiency and iron-deficiency anemia
         pp 635-670
           Laboratory findings
           pp 656-660
     Process: 134.1: Dependence fact from authority source
     By: GW Gilroy RPh-SP: 1/22/79
     Documentation not audited
```

Figure 4. Fact in expanded text form.

> "The ingredients needed to offer computer-based health-care guidance are now all in place. Knowledge is captured in atomic form in the facts. Facts are organized using a directed network. The network facilitates understanding and use of the facts to offer guidance. Building of this guidance capability is done in an explicit and rigorous fashion. Each step in the process of transforming knowledge to facts to guidance is documented and connected in electronic form to make both guidance and knowledge maintainable and adaptable."

13. The Master Library and Instances

The library system incorporated ideas outlined in the Conway report on the National Library of Displays. There were three classes of information in the generic library:

1. generic information that applied to all instances;
2. local information that applied to only one instance;
3. protocol information that may apply to several instances, but is designed and built with a specific set of values in mind.

Generic and protocol information were integrated into PROMIS by the library system. The generic library was responsible for maintaining all generic information and some protocol information. All other information was to be maintained locally by the instance that built and used it. All instance-specific data was identified in the PROMIS frame and table library, and procedures were implemented to allow the updating of only the generic information.

14. PROMIS Returns to a General Medical Ward

In December of 1976, the newly implemented PROMIS became operational on a 20 bed general medical ward (Brown III) in the Medical Center Hospital. The Brown III site was linked to the pharmacy, clinical laboratory and the radiology departments.

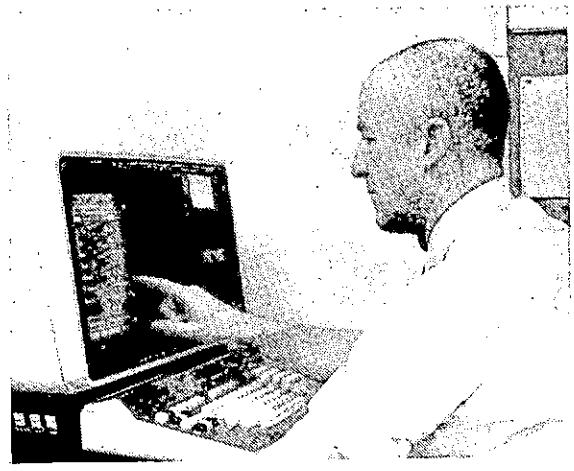
Functionally the new PROMIS was a major improvement over the CDC system. It was easier to use and to maintain. It had expanded functions and was more responsive and reliable. Until September of 1981 when PROMIS Laboratory left the University of Vermont and became PROMIS Information Systems, Inc., there were a steady stream of improvements and new functions. Below is a description of some of the features of the 1980 PROMIS system.

14.1. Addition of administrative functions

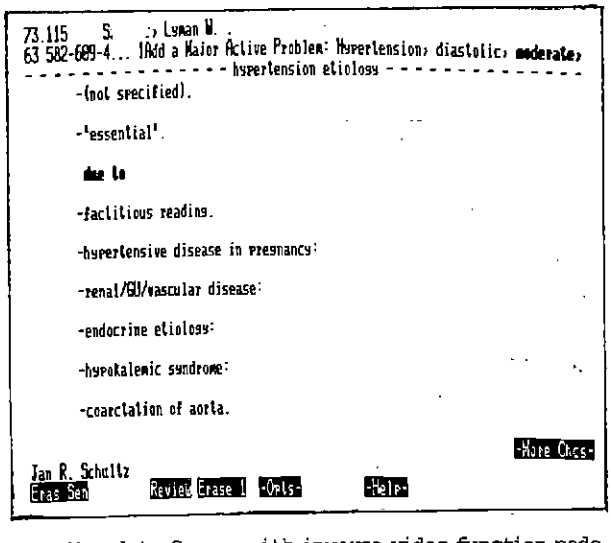
The administrative functions that are the basis for most computer based medical systems were not initially integrated into PROMIS. LeMay, along with other members of the PROMIS computer group, rewrote an existing administrative package to operate in conjunction with the electronic patient record. The resulting administrative system included patient billing, accounts receivable, accounts payable, and general ledger functions. All manual input was entered at a terminal. Admission, discharge, transfer, and charge data were generated automatically as a byproduct of use of the electronic record. Input to the general ledger from patient billing, accounts receivable, and accounts payable was generated automatically.

14.2 Networking was made routine

A two-node configuration was used routinely for PROMIS and system development; additionally, heavy simulated loads were applied to three nodes in order to validate the effects of contention for the basic network resources. Experiments were performed placing all terminals on one node and all medical records on another; this forced internode activity for each storage and retrieval operation. Under these conditions the user perceived little if any performance degradation due to the network.



Larry Weed using the Megadata Terminal.



Megadata Screen with inverse video function pads.

14.3. Population studies were added to PROMIS

The ability to interrogate the electronic patient records as a group for patient care or clinical research was provided by the population study system, developed by Stephen Reynolds. Population studies attempted to answer questions a clinical investigator might ask of the patient records. The population study system enabled three types of interrogation:

1)	Type of question:	Who has a specified combination of attributes?
	Form of result:	List of patients
	Example:	Who is receiving gentamicin?
2)	Type of question:	What are the results of a set of specified procedures?
	Form of result:	Array of values
	Example:	What is the standard deviation of BUN in patient group Y?
3)	Type of question:	What problems appear on the problem lists of patients in a specified group?
	Form of result:	Ordered list of problems
	Example:	What is the incidence of all problems identified in patients in group Z who have diabetes?

Peer Standards Review Organization personnel at the hospital performed audits using the population study tools, the results of which were submitted to the hospital administration and medical board. Questions were similar to:

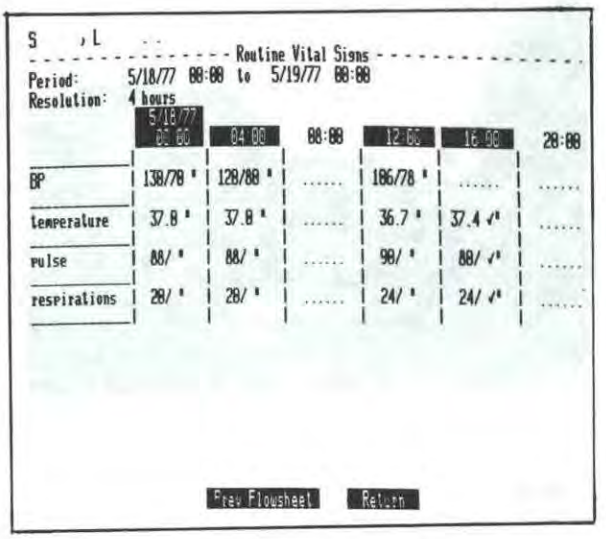
IF (date of serum creatinine) MINUS (date of digoxin) GREATER THAN 2 OR (date of digoxin) MINUS (date of serum creatinine) GREATER THAN 2
THEN "Serum creatinine not ordered in time"

The result of this query would be a list of patients for whom the assessment "Serum creatinine not ordered in time", as defined by the Boolean expression, was correct.

The processing and display of a patient set based upon the simple presence or absence of attributes was done while the user waited - usually within ten seconds or less; a patient set based on attributes qualified by time or other variables took longer depending on the complexity of the question and the number of patient records processed.

It was also possible to pass data from the electronic patient records to the MINITAB II statistical package (developed by the National Bureau of Standards and modified at the University of Pennsylvania) for statistical analysis. MINITAB was implemented with the commands as choices on frames, allowing easy, rapid, controlled and structured access.

All patient attributes within the electronic records that were used for population studies had to be coded. Complex relationships within one procedure report could not be captured. A coding mechanism was needed to parallel the text string seen by the user. The development of the code string, a tree-structured, frame-driven encoding mechanism, would have to wait for the next application design iteration.



Example of vital signs flowsheet for a single day.

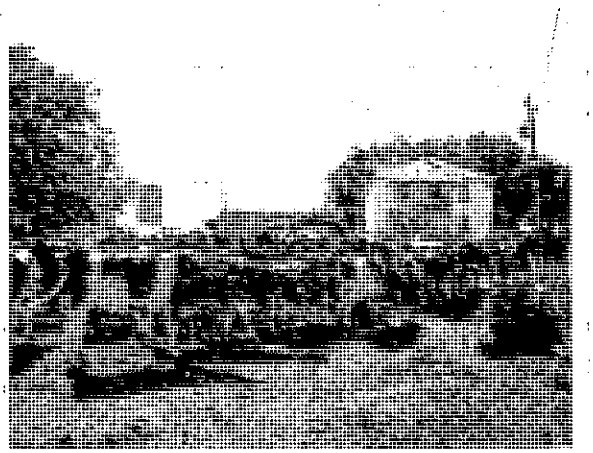
14.4. Patient Lifetime Records were developed

The structure of the electronic record was expanded to allow multiple inpatient admissions and outpatient visits within one record. The encounter was the structuring mechanism for the patient's lifetime record. Each inpatient admission or outpatient visit was a new encounter. The user could specify any encounter (or all encounters) from which to retrieve patient data.

14.5. Communications System Use

The MEGADATA touch-screen terminals, as well as printers and other peripherals, were connected to the cpu with a high speed communications bus. The data communications lines could be extended up to 20 miles (32 km) by cable and further extended with microwave links. A very high frequency (VHF) 2-way coaxial bus was employed to provide many

inexpensive interconnect ports. Full-duplex modems operated at 307,200 bits per second. Programmable controllers managed peripheral devices and operated at the remote interconnect ports. The line control protocol was a simple polling protocol. Each remote device was assigned a unique poll address. Because of the line protocol and the nature of the bus, users were free to move terminals and other peripherals from one location to another and continue without any access (sign-on) requirements. (Wanner 1978). The bus was extended with a microwave link to the Grand Isle Medical Clinic 17 miles from the Hospital.



The Grand Isle Clinic. Find the microwave dish?

15. Conclusion

15.1 The Human Interface

The combination of a response rate of less than .25 second and the large data base of frames allows users to navigate through the network of frames very rapidly, accomplishing their tasks without using the keyboard. The interaction rate is very high since a single selection results in the presentation of a full screen of new text. A complete thought can be communicated in a single touch. The data base available to a user is a combination of electronic records generated from past selections and of frame and table knowledge. The user can operate from a universe larger than what can be remembered and has only to recognize the correct selection (and not recall it). These features provide for a unique and very effective human-computer interface.

The building of the data base of frame and table knowledge is an intellectually deep task and is time consuming. The PROMIS frame and table library took over 200 person-years of effort to build. This type of interface can be successful if the cost of developing and maintaining the knowledge base used by the interface can be spread across many users.

In order for the interface to work, it requires an extensive knowledge base as well as a different relationship to the computing resources it uses. For PROMIS, the interface is the computation and not a means to get to a computation. As computing power gets less expensive, this interface becomes more cost effective.

Robertson, Newell, and Ramakrishna discuss the PROMIS/ZOG type of interface and claim that this "type of interface is a preferred mode of man-machine interaction, even over the use of natural language dialogue with the computer in the role of intelligent agent.

"There are two polar views about how to structure man-machine interaction. One is the computer as tool. ... Control remains with the user. The other view is the computer as intelligent assistant. In this view one wishes to make the computer more intelligent and communication with it more natural ... Precisely what an intelligent assistant is supposed to provide is freedom (of the user) from the effort of understanding. Put one other way, delegation requires an act of faith.

"ZOG is an evolution in the tool direction. It seeks to produce a transparent device which, in itself, has no intelligence at all, but is immensely responsive to the user. It seeks to do this in the arena where we normally expect to use natural language, namely, dealing with large bodies of knowledge. Indeed ZOG uses natural language for its output (though arranged in a sort of spatial dialogue), for the user has good devices for assimilating it. However, its own internal structure, which governs what it says and when it chooses to say it, is completely open to examination by the user. (Robertson 1977)

15.2. Application Development

In October of 1981, PROMIS Laboratory left the University of Vermont and formed PROMIS Information Systems, Inc. We realized that the potential for the PROMIS technology was broader than medicine alone, and that work was needed on the application development software. From the very beginning of our development in 1967, the Human Interface

Program interpreted the frame in the process of bringing up the next display. It was table driven since the frame (the programs table) defined how the program was to operate. Other parts of the system were not table driven. The frame editor has been table driven since 1978, but the storage and retrieval programs had imbedded within them the logic of the Problem Oriented Medical Record and could handle only one type of record structure. We have rewritten the application subsystems so that they are all table driven. This allows the development of systems as large and complex as the PROMIS medical application without writing PPL programs.

Application development without writing programs makes programmers more efficient by a factor of between 10 and 100. What we have developed is not a programmer-less system, but a set of tools that makes programmers more effective. The tool kit uses the concepts of programming to accomplish the tasks of programming, but the writing of procedural programs is not required.

15.3. The Future

When we began our work, a touch-screen terminal operating over telephone lines at 2400 bits per second cost over $20,000. Today, a similar terminal could be purchased off-the-shelf for under $3,000. The personal computer I'm using to write this paper has more central memory in it then the CDC 1700 system used for the initial development and the personal computer's central memory cost 200 times less. The technology is available now to make the type of interface and systems described in this paper cost effective and widely available.

The PROMIS system is alive and kicking. It is still the most advanced medical information system in existence and the only one to manage a fully electronic medical record. It has yet to be used as the information system for a total hospital, but its time is coming.

Acknowledgments

This paper represents the efforts of many individuals in medicine and in computer science, both within and outside the past PROMIS Laboratory. I would like to acknowledge the group from PROMIS Laboratory who have stayed together to keep these ideas alive. This work was funded primarily by the National Center for Health Services Research, Department of Health, Education, and Welfare.

References

(Hertzberg, Schultz, Wanner 1980) Hertzberg, R.Y., Schultz, J.R., Wanner, J.F.
The PROMIS Network
Computer Networks 4, 1980 pp. 215 - 228

(Parnas 1972) Parnas, D.L.
A Technique for Software Module Specification with Examples
Comm. ACM 15(5), pp.330-336 (May 1972)

PROMIS Laboratory Staff
The Representation of Medical Knowledge
Proc. Second Annual Symposium on Computer Applications in Medical Care, Washington, DC, Nov. 1978, pp 368-400.

(Robertson 1977) Robertson, G., Newell, A., Ramakrishna, K.
ZOG: A Man-Machine Communication Philosophy
Department of Computer Science, Carnegie-Mellon University, August 4, 1977.

(Schultz, Davis 79) Schultz, J.R., Davis, L.
The Technology of PROMIS
Proc. of IEEE, Vol. 67, 9, Sept. 1979.

(Schultz, Cantrill, Morgan 1971) Schultz, J.R., Cantrill, S.V., Morgan, K.G.
An Initial operational problem oriented medical record system - for storage, manipulation and retrieval of medical data
AFIPS - Conference Proceedings, Vol 38, 1971.

(Walton, Holland, Wolf 1979) Walton, P.L., Holland, R.R., Wolf, L.L.
Medical Guidance and PROMIS
IEEE, Computer, November, 1979, pp 19 - 27.

(Wanner 78) Wanner, J.F.
Wideband Communication System Improves Response Time
Computer Design, Dec. 1978, pp. 85 - 91.

(Weed 64) Weed, Lawrence L., MD.
Medical Records, Patient Care and Medical Education
Ir. J. Med. Sc., June 1964, pp. 271 - 282.

(Weed 68) Weed, Lawrence L., MD.

Medical Records that Guide and Teach
New England Journal of Medicine, 278:593-600, 652-657
(March 14, 21), 1968.

(Weed 69) Weed, Lawrence L., MD.
Medical Records, Medical Education and Patient Care
Case Western Reserve Press, Cleveland, Ohio, 1969.

User Technology: From Pointing to Pondering

Stuart K. Card and Thomas P. Moran
Xerox Palo Alto Research Center

From its beginning, the technology of personal workstations has been driven by visions of a future in which people would work in intimate partnership with computer systems on significant intellectual tasks. These visions have been expressed in various forms: Memex (Bush, 1945), Man-Machine Symbiosis (Licklider, 1960), NLS (Engelbart, 1963), Dynabook (Kay, 1977), and others.

The tight coupling between human and computer required by these visions necessitated advances in the ways humans and computers interact. These advances have slowly begun to accumulate into what might be called a *user technology*. This user technology includes hardware and software techniques for building effective user interfaces: bitmapped displays, menus, pointing devices, "modeless" command languages, animation, and interface metaphors. But it must include a technical understanding of the user himself and of the nature of human-computer interaction. This latter part, the scientific base of user technology, is necessary in order to understand why interaction techniques are (or are not) successful, to help us invent new techniques, and to pave the way for machines that aid humans in performing significant intellectual tasks.

In this paper, we trace some of the history of our understanding of users and their interaction with workstations—the personal part of personal workstations. In keeping with the spirit of other papers at this conference, we have centered this review around our own experiences, perspectives, and work and have not attempted a complete history of the field. In concentrating on our own work, we do not wish to mimimize the importance of others' work; we simply want to tell our own story. Our focus is on what we have learned about users in our years of studying them and how we see our findings relating to the original visions of the personal workstation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0183 $00.75

1. The Vision of an Applied User Psychology

The opportunity to tackle a science of the user brought us to PARC in 1974 (collaborating with Allen Newell, as consultant). As other PARC researchers were beginning to pursue the vision of highly graphic, interactive, network-based personal workstations, we were following a vision of our own. The idea was to draw concepts from cognitive psychology and artificial intelligence to create an applied cognitive science of the user. We called our project the Applied Information-processing Psychology Project (AIP). A 1971 memo by Allen Newell proposing this project to PARC stated the basic argument this way:

> (1) There is emerging a psychology of cognitive behavior that will permit calculation of behavior in new situations and with new humans. . . .
>
> (2) Several of the tasks that are central to the activities of computing—programming, debugging, etc.—are tasks that appear to be within the early scope of this emerging theory.
>
> (3) Computer science in general is extremely one-sided (for understandable reasons) in the treatment of its phenomena: almost no effort goes into understanding the nature of the human user. . . .
>
> (4) There is a substantial payoff (in dollars) to be had by really designing systems with detailed understanding of the way the human must process the information attendant thereto.

In 1974, we were in the position of having to create a new field. Psychological theories and methodologies held the promise of being able to represent and manage complex cognitive tasks, but the only body of research pertaining to human-computer interaction was in the field of human factors, where studies were largely empirical and evaluative, concentrating mostly on sensory-motor questions like the best shape for a switch. Human-computer interaction, involving two active agents each capable of initiating an exchange of information, inherently involves human

cognitive processing. Our vision was to create a science of the user rooted in cognitive theory. But we also wanted the science to be practical, providing the system designer with the conceptual tools to think about the key characteristics of the user and the calculational tools to take account of the user's behavior.

To simplify a rather complex history for the purposes of this paper, we will narrate just a few strands of what unfolded. Each strand is organized around a particular aspect of the interface between the user, the system, and the task. The strands are organized into a sequence of ever-broader interfaces, from the physical interaction of the user with devices and displays to the symbiotic interaction of the user with the system while grappling with complex intellectual tasks—that is, from pointing to pondering:

> *The physical interface:* The user interacts with a system by means of physical input devices, such as a keyboard and a mouse, and output devices, such as a high-quality graphical display.
>
> *The cognitive interface:* The user has certain characteristics as an information processor, such as a limited working memory, that together with the goals he is trying to achieve determine his behavior.
>
> *The conceptual interface:* The computer system is also a complex information processing mechanism, and the the user needs to have some kind of mental model of it in order to effectively interact with it.
>
> *The task interface:* Systems are designed to help their users do tasks, not only small, routine tasks, but also larger, difficult intellectual tasks that are the object of the grand visions of personal workstations.

The story within each strand of our work is chronological; but the first three strands, representing our previous work, overlap in time. The last strand represents our current work.

2. The Physical Interface: Pointing

Bill English, who had among other thing engineered and built the mouse for Doug Engelbart (English, Engelbart, and Berman, 1967), was one of the people who migrated to PARC when it was founded. We worked for English when we first came to PARC. Engelbart and English had always considered the mouse to be an interim device, and English wanted to see if it was possible to invent other devices that would improve on its speed. In particular, he was interested in whether one could put a key on the keyboard that would sense force and direction without requiring the user to remove his hands from the keyboard. He had a number of devices built, and we helped design an experiment to test them. The results (Figure 1) showed that none of the devices tested improved on the mouse either for speed or for error rate (Card, English, & Burr, 1978).

This was a typical human factors experiment: we compared a set of systems to determine which one performed best. But this direct empirical comparison between devices was just the sort of methodology of human factors testing that we wanted to improve: we wanted to understand the reasons why the results came out the way they did. We therefore made mathematical models of each device and tested them against the data until we found models that fit. The model for the mouse was particularly instructive. The mouse was best modeled by a version of Fitts's Law:

$$Movement\ Time = Constant + .1 \log_2(D/S + .05)\ \text{sec},$$

		Movement Time for Non-Error Trials						Error Rate	
		Homing Time		Positioning Time		Total Time			
Device	Trials N	M (sec)	SD (sec)	M (sec)	SD (sec)	M (sec)	SD (sec)	M	SD
Mouse	1973	.36	.13	1.29	.42	1.66	.48	5%	22%
Joystick	1869	.26	.11	1.57	.54	1.83	.57	11%	31%
Step Keys	1813	.21	.30	2.31	1.52	2.51	1.64	13%	33%
Text Keys	1877	.32	.61	1.95	1.30	2.26	1.70	9%	28%

Figure 1. Overall pointing times for all devices. Homing Time is the time to move the hand from the keyboard to the device, Positioning Time is the time to move the cursor to the target. Times are based on a standardized set of distances and target widths. Averages are computed on the basis of four users x 600 trials/users. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.4) with the permission of Lawrence Erlbaum Associates, Inc.*

where D is the distance the hand moves to the target and S is the target width (see Figure 2). The significance of this result is that this is the same law that describes movement time for the hand alone, with about the same constant of proportionality. The limiting factor in moving the mouse, therefore, is not in the mouse, but in the eye-hand coordinate system itself. That, in turn, means the mouse is nearly optimal, at least with respect to the set of muscles used. Therefore, designing a device that is faster than the mouse would be difficult.

Here was a prototypical example of the kind of theory we wanted to build—a model precise enough to enable designers to perform back-of-the-envelope calculations, a model that identified key constraints in the design space for pointing devices. With this model we could be sure that the data giving comparisons among devices would probably generalize to new situations, because we knew the main factors that governed the results. Furthermore, the model provided guidance for interface designers: make distant buttons large, for example. These studies were heavily used in the debate within Xerox that led to the decision to depart from tradition by including a mouse with the new Star product.

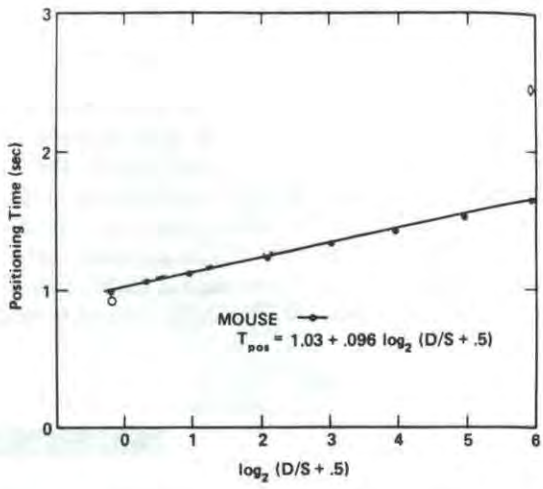


Figure 2. Positioning time for the mouse as a function of Fitts's Index of Difficulty. The fit of positioning time data to the straight line in the figure shows that time for the mouse is governed by Fitts's Law (is proportional to Fitts's Index of Difficulty). The constant of proportionality, .096 sec/bit (= 10.4 bits/sec), is approximately the same value as for pointing with the hand alone. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.8) with the permission of Lawrence Erlbaum Associates, Inc.*

3. The Cognitive Interface: Cognitive Skill

The Model Human Processor

An interesting result of the mouse study was the way an evaluation of pointing devices led to a consideration of human information processing characteristics. This experience pointed to the need for an engineering model of users that would summarize such characteristics. We knew of many phenomena scattered in the literature of psychology, such as Fitts's law, that would be helpful for system design.

To someone who is not a specialist, such as a designer, this literature appears disorganized and contradictory. Psychologists love to split hairs and find small contradictions in published models. The robust but approximate generalizations that might be made to work for engineering tend to get trampled in the debates. Although we had the notion of such a model from about 1974, it wasn't until 1982, when we were nearing the completion of our book, that we were able to formulate it. The model, called the *Model Human Processor*, was inspired by the processors, memories, and switches (PMS) notation of Bell and Newell (1971) for describing the architecture of computing systems. It was a simplified architecture of the user, described in terms of three processors, four memories, 19 parameters of these, and 10 principles of operation (Figure 3).

As an example of the sort of calculation that can be done with the Model Human Processor, consider the case where a programmer is programming a video game version of billiards. He needs to know how long he has after the collision of two balls to compute the balls new trajectory before the illusion of causality breaks down. The Model Human Processor tells us that every event that occurs within 100 msec will be perceived as a single event, so a rough estimate is 100 msec. But it also recognizes that second-order phenomena can change this number, so it also supplies a range of uncertainty for this number (in this instance, 50 to 200 msec).

In this case, we can say that if the programmer can make the balls move within the 50 msec lower bound for this parameter, then it is pretty certain that users will perceive the collision as the cause of the balls change in direction, regardless of secondary effects such as brightness or contrast ratio of the screen.

The Model Human Processor can be used to compute predictions about human performance: how fast people can read, how fast they can scribble, the effect of different abbreviation schemes on memory error, and so on. Derivations based on the Model Human Processor were used to set the maximum velocity of the mouse on the workstation for the Xerox Star.

Text Editing and Cognitive Skill

The Model Human Processor also contains, as one of its principles of operation, Herbert Simon's bounded rationality principle. Our version of this principle may be stated:

> *A person acts so as to attain his goals through rational action, given the structure of the task, his*

inputs of information, and bounded by limitations upon his knowledge and processing ability. (Card, Moran, & Newell, 1983).

That is, in order to predict a person's behavior, one needs to analyze the task he is trying to do, because the person will simply do what is rational to accomplish the task, constrained by limitations in his knowledge and ability to process information. Studies of how people solve problems had shown (Newell & Simon, 1972) that their behavior could be modeled as a search through a space of states of the problem, a *problem space.* We expected that in studies of human-computer interaction we would find users searching through problem spaces to accomplish goals, trying various solutions, backing off and taking other tacks when they ran into trouble.

Two early decisions we made led us to results which were contrary to our expectations. Although we had come to PARC initially with the intention of studying computer programming, we decided once we arrived that there were strategic advantages to studying text editing. A second, tactical, decision was to work with expert subjects rather than novices, in order to have more stable behavior to analyze. Further, to make sure that all our subjects had the

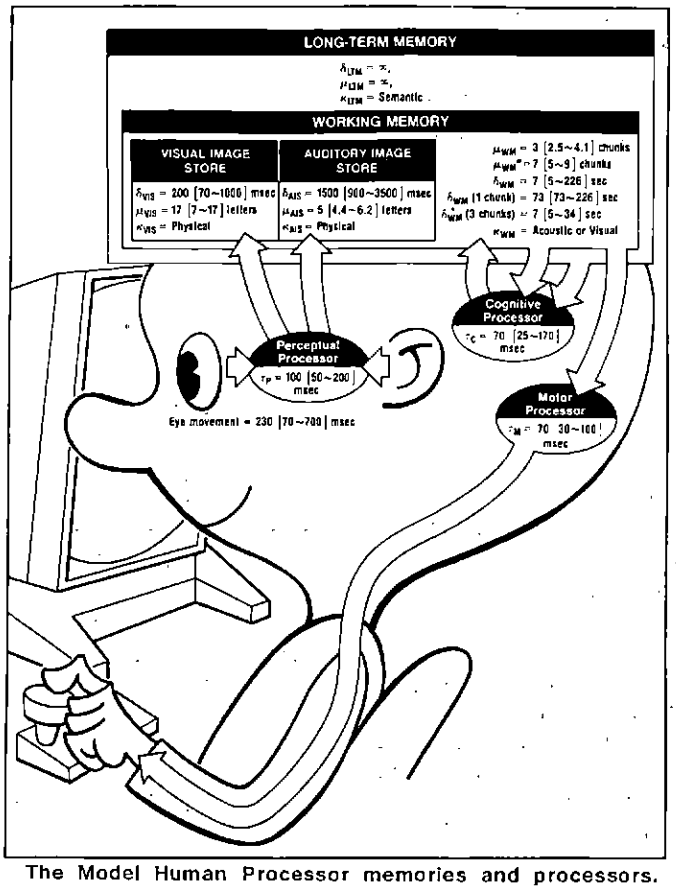


Figure 3. The Model Human Processor memories and processors. Depicted schematically in the figure are the memories, processors, and constants used for making simple computations. The basic architecture of the model can be summarized thus: Sensory information flows into Working Memory through the Perceptual Processor. Working Memory consists of activated chunks in Long-Term Memory. The basic principle of operation of the Model Human Processor is the Recognize-Act Cycle of the Cognitive Processor: On each cycle of the Cognitive Processor, the contents of Working Memory initiate actions associatively linked to them in Long-Term Memory; these actions in turn modify the contents of Working Memory. The Motor Processor is set in motion through activiation of chunks in Working Memory.

Predictions are made using time constants from the figure and a set of associated Principles of Operation: (P0) The Recognize-Act cycle of the Cognitive Processor; (P1) The Variable Perceptual Processor Rate Principle; (P2) The Encoding Specificity Principle; (P3) The Discrimination-Principle; (P4) The Variable Cognitive Processor Rate Principle; (P5) Fitts's Law; (P6) The Power Law of Practice; (P7) The Psychological Uncertainty Prinicple; (P8) The Rationality Principle; and (P9) The Problem Space Principle. *Reprinted from Card, Moran, & Newell (1983, Fig. 2.1.) with the permission of Lawrence Erlbaum Associates, Inc.*

same goals in mind, we presented them with a fairly explicit task—the "manuscript editing task"—which required them to work from a marked-up manuscript, making the changes explicitly indicated.

When we analyzed videotaped protocols of our subjects doing this task, we were surprised to find little of the search behavior of problem solving that we had expected. Subjects simply looked at the tasks and did them. The tasks were not problematic. The subjects had done similar tasks many times before, and had built up a large repertoire of methods that could be applied to the tasks. This wasn't problem solving; we came to call it *cognitive skill* and set out to build models to characterize and predict this mode of behavior.

We applied our theories of human information processing to the kind of specific skills necessary for the text-editing task. The result was a class of models in which the user's cognitive structure consists of four components: (1) a set of familiar Goals that the user would recognize when faced with a specific task; (2) a set of primitive Operators (actions) that the user was skilled at performing and could deploy whenever necessary; (3) a set of Methods, consisting of "compiled" sequences of subgoals and operators, that the user could use to attain his goals; and (4) a set of Selection rules that enable the user to choose among competing methods for goals. We call a model specified by these components a *GOMS model* (Figure 4 shows an example of one). Together, these components constitute the user's cognitive skills for performing tasks. If a user has enough knowledge of this kind, it isn't necessary to use problem-solving strategies. All that is needed is to examine the task, characterize it in terms of a specific goal, select the appropriate method, and then execute it.

In order to test our GOMS models, we ran a set of experiments to determine whether we could explicitly specify this kind of GOMS knowledge and thus predict what users would actually do. We also wanted to learn how the degree of resolution—the grain of specified detail of such a model—affects the degree of accuracy of predictions based on the model. We expected fine-grained models to yield increased accuracy. Knowing that it takes a lot more work to develop a fine-grained model, a practical question related to the applicability of our models was whether the additional work of constructing a fine-grained model was worth the effort.

To find out, we built a family of models in the GOMS framework that characterized the behavior of users of a specific computer text-editing system and ran experiments to test our predictions from these models. We were surprised to discover that fine-grained models did not yield a worthwhile or even a significant increase in prediction accuracy (Card, Moran, & Newell, 1976, 1980a). More felicitous, practically speaking, was our discovery that even the crude models seemed to capture and predict behavior fairly well. These properties suggested that the GOMS model could be turned into the kind of engineering tool that a designer could use to model and predict skilled user behavior in computer-mediated tasks. That was what we proceeded to try next.

KEYSTROKE LEVEL

```
Model K2:
GOAL: EDIT-MANUSCRIPT
. GOAL: EDIT-UNIT-TASK                                  . repeat until no more unit tasks
. . GOAL: ACQUIRE-UNIT-TASK                             . . if task not remembered
. . . GOAL: TURN-PAGE* (see below)                      . . . if at end of manuscript page
. . . GOAL: GET-FROM-MANUSCRIPT*
. . GOAL: EXECUTE-UNIT-TASK                             . . if an edit task was found
. . . GOAL: LOCATE-LINE                                 . . . if task not on current line
. . . . CHOOSE-COMMAND
. . . . [select  GOAL: USE-QS-METHOD
. . . . .              GOAL: SPECIFY-COMMAND*
. . . . .              GOAL: SPECIFY-ARG*
. . . . .        GOAL: USE-LF-METHOD
. . . . .              GOAL: SPECIFY-COMMAND*]          . . . . repeat until at line
. . . . GOAL: VERIFY-LOC*
. . . GOAL: MODIFY-TEXT
. . . . CHOOSE-COMMAND
. . . . [select  GOAL: USE-S-COMMAND
. . . . .              GOAL: SPECIFY-COMMAND*
. . . . .              GOAL: SPECIFY-ARG*
. . . . .              GOAL: SPECIFY-ARG*
. . . . .        GOAL: USE-M-COMMAND
. . . . .              GOAL: SPECIFY-COMMAND*
. . . . .              GOAL: SPECIFY-COMMAND*          . . . . repeat until at text
. . . . .              GOAL: SPECIFY-ARG*
. . . . .              GOAL: SPECIFY-COMMAND*]
. . . . GOAL: VERIFY-EDIT*

* Expansion of goals appearing several times:
        GOAL: TURN-PAGE
        . LOOK-AT-MANUSCRIPT                            . repeat twice
        . ACTION
        . . MOVE-HAND                                   . repeat twice
        . . TURN-PAGE
        GOAL: GET-FROM-MANUSCRIPT
        . LOOK-AT-MANUSCRIPT
        . SEARCH-FOR
        . LOOK-AT-DISPLAY                               . optional
        GOAL: SPECIFY-COMMAND
        . GOAL: GET-FROM-MANUSCRIPT*                    . if not already selected
        . CHOOSE-COMMAND                                . if not already selected
        . GOAL: TYPE-STRING*
        GOAL: SPECIFY-ARG
        . GOAL: GET-FROM-MANUSCRIPT*                    . optional
        . CHOOSE-ARG
        . GOAL: TYPE-STRING*
        GOAL: VERIFY
        . LOOK-AT-DISPLAY
        . GOAL: GET-FROM-MANUSCRIPT*                    . optional
        . COMPARE
        GOAL: TYPE-STRING
        . HOME                                          . optional
        . LOOK-AT-KEYBOARD                              . optional
        . LOOK-AT-DISPLAY                               . optional
        . TYPE-STRING
```

Figure 4. GOMS Model K2 for the text editor POET. This model has a grain of about 0.5 sec/operator. The user is using a line-oriented editor to make changes to a computer file previously marked on a paper manuscript. *Reprinted from Card, Moran, & Newell (1983, Fig. 5.12) with the permission of Lawrence Erlbaum Associates, publishers.*

The Keystroke-Level Model

In simplifying the GOMS models into an engineering model that we could hand to a designer, we constructed an idealized prediction problem:

> *Given a task (possibly involving several subtasks), the command language of a system, the motor skill parameters of the user, the response time parameters of the system, and the method used for the task, predict how long an expert user will take to execute the task using the system, providing he uses the method without error.* (Card, Moran, & Newell, 1980b)

The Keystroke-Level Model enables a system designer to make such predictions with a "back-of-the-envelope" style of calculation.

To do the calculation, the designer codes the method a user employs to do a task in terms of a set of operations derived from one of the fine grain GOMS models (Figure 5). In this simplified model, all keystrokes are assumed to take a constant amount of time; and pointing with the mouse is also assumed to take a constant amount of time. Mental activity by the user is reduced to a single generic mental preparation operation, and rules governing when it will occur are provided.

A typical use of the model to analyze a method is given in Figure 6. This method is one way in which a hypothetical text-editor could be used to replace a word recently typed by the user. Each action of the method is described in terms of the operators of the model, then a time for the method is computed. In this case, the method is expected to take the same amount of time regardless of how many words back the word to be replaced is located. Figure 7 shows the expected time for performing this methods and two other methods available in the editor as a function of the number of words back the to-be-changed word is located. It can be seen that the time profiles of the three methods are quite different and that each is fastest at different times.

Operator	Description and Remarks	Time (sec)
K	PRESS KEY OR BUTTON. Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	.08
	Good typist (80 wpm)	.12
	Average skilled typist (55 wpm)	.20
	Average non-secretary typist (40 wpm)	.28
	Typing random letters	.50
	Typing complex codes	.75
	Worst typist (unfamiliar with keyboard)	1.20
P	POINT WITH MOUSE TO TARGET ON A DISPLAY. The time to point varies with distance and target size according to Fitts's Law, ranging from .8 to 1.5 sec, with 1.1 being an average. This operator does *not* include the (.2 sec) button press that often follows. Mouse pointing time is also a good estimate for other efficient analogue pointing devices, such as joysticks (see Chapter 7).	1.10
H	HOME HAND(S) ON KEYBOARD OR OTHER DEVICE.	.40
D(n_D,l_D)	DRAW n_D STRAIGHT-LINE SEGMENTS OF TOTAL LENGTH l_D CM. This is a very restricted operator; it assumes that drawing is done with the mouse on a system that constrains all lines to fall on a square .56 cm grid. Users vary in their drawing skill; the time given is an average value.	$.9n_D + .16l_D$
M	MENTALLY PREPARE.	1.35
R(t)	RESPONSE BY SYSTEM. Different commands require different response times. The response time is counted only if it causes the user to wait.	t

Figure 5. The operators of the Keystroke-Level Model. The figure lists the operators needed to analyze user interface methods and to make calculations of user performance with these methods. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.1) with the permission of Lawrence Erlbaum Associates, Inc.*

Even though our GOMS study suggested that simple models could be effective, and even though the Keystroke-Level Model was a careful simplification of one of the GOMS models, we felt that it was necessary to rigorously test the explicit performance assumptions of the model. To validate the Keystroke-Level Model, we ran a large set of experiments in which people performed tasks with text editors, graphics editors, and executive

Method R (Replace):	
Terminate type-in mode	**MK**[ESC]
Point to target word and select it	**H**[mouse] **P**[word] **K**[YELLOW]
Call Replace command	**H**[keyboard] **MK**[R]
Type new word	**4.5K**[word]
Terminate Replace command	**MK**[ESC]
Point to last input word and select it	**H**[mouse] **P**[word] **K**[YELLOW]
Re-enter type-in mode	**H**[keyboard] **MK**[I]

$$T_{execute} = 4t_M + 10.5t_K + 4t_H + 2t_P$$
$$= 12.1 \text{ sec}.$$

Figure 6. Encoding of Method R. Use of the Keystroke-Level Model to describe one possible method in a mouse-based display editor for replacing a word previously mistyped. It is assumed the word is still visible on the screen. *Reprinted from Card, Moran, & Newell (1983, p. 289) with the permission of Lawrence Erlbaum Associates, Inc.*

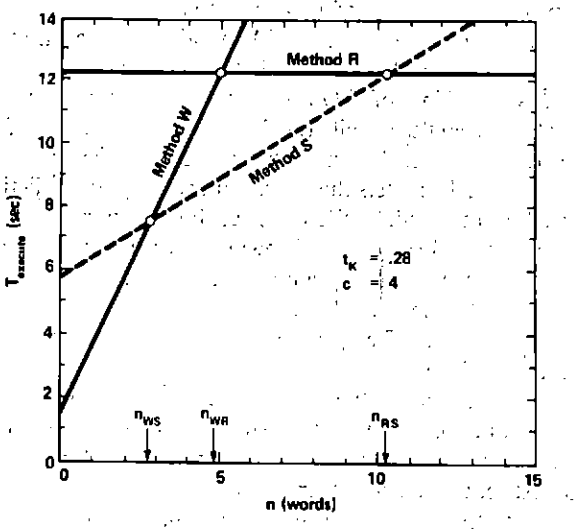


Figure 7. Execution time of three methods for the misspelled-word task as a function of *n*. This figure shows the use of the Keystroke-Level Model for the parametric comparison of three different methods for accomplishing the same goal. In this case each method appears to be superior in a certain range of *n*. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.12a) with the permission of Lawrence Erlbaum Associates, Inc.*

subsystems. The results (Figure 8) showed a good fit between predictions derived from the model and observed times required for the tasks. Therefore, our simple approximative model would be of some use in making practical time estimates.

The Keystroke-Level Model has actually proved useful in real system design. One application was the determination of the number of buttons on the mouse for the Xerox Star product. Several schemes for selecting text in the Star text editor were proposed. These schemes required different numbers of buttons. The goal was to make a mouse with the smallest number of buttons possible, so that it was easy to learn to operate. Experiments to test the schemes were reasonably easy to run with novice subjects. Everyone is a novice subject on a new system, and being a novice doesn't require training. But understanding how well the schemes would work for expert users (which most users would be for most of their time on the system) was expensive, because a long time would have to be spent training the users. The solution was to run experiments for novices and to use the Keystroke-Level Model for predicting expert performance.

The Keystroke-Level Model allowed us to carry through, at least in regard to a very specific kind of behavior, part of our original vision of packaging psychological knowledge into a model that designers can use to calculate user performance with a variety of interactive computer systems.

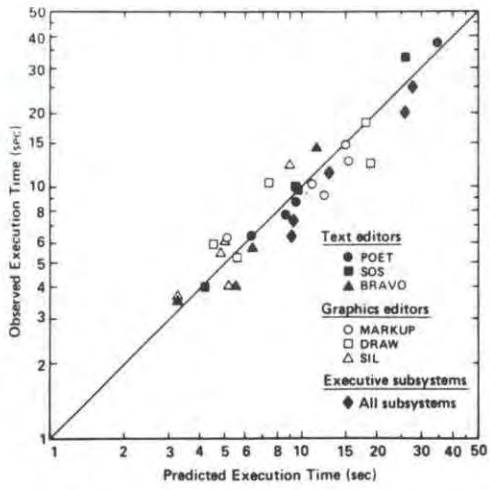


Figure 8. Predicted vs. observed execution times in tests of the Keystroke-Level Model. Predicted execution times were calculated from the Keystroke-Level Model. Observed execution times come from empirical observation. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.6) with the permission of Lawrence Erlbaum Associates, Inc.*

The Unit Task

Not all skill characteristics can be reduced to simple counting, however. In studying text editing, we observed a characteristic of cognitive skill that is fundamentally cognitive, namely the organization of user behavior into short, quasi-independent tasks, which we call *unit-tasks.* Figure 9 shows a typical timeline of user behavior, in this case from a protocol of an electrical engineer using a graphic CAD system to design a VLSI circuit. The figure shows that the behavior is divided into chunks, each consisting of a a few seconds of pause (to formulate a task to do) followed by a few seconds of activity (to execute the task). Each think-execute chunk of behavior is a unit task.

The unit task structure of cognitive skill is interesting because the performance limitations of the user show through the purely rational organization of his behavior. The most significant reason why the unit task breakdown of behavior arises is because of limitations in the user's working memory. If the user can manage input and output streams in his working memory, then the user's behavior will have a continuous structure, as in touch typing from a manuscript. But, when conditions on the inputs and outputs do not allow this, then behavior must be structured into a series of unit tasks.

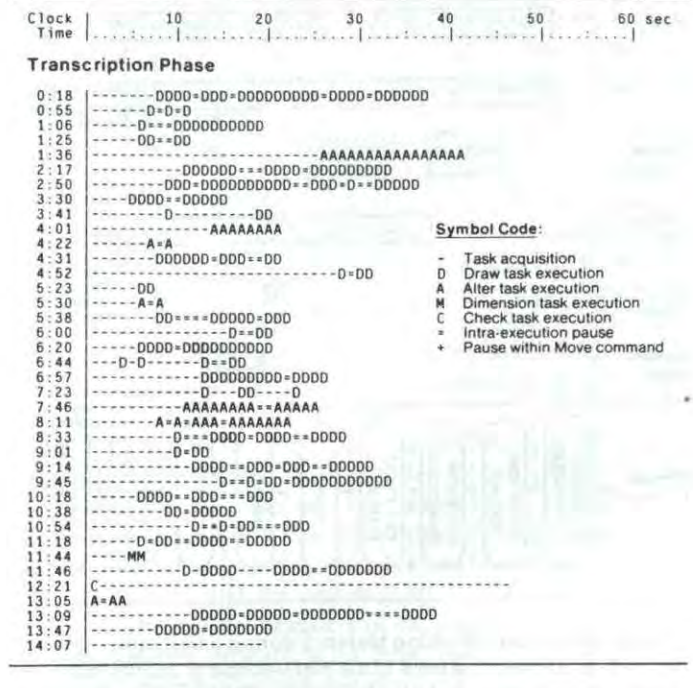


Figure 9. Time line representing the user's behavior sequence in a VLSI design session. Each single-character symbol represents one second of behavior. The symbol sequence begins on a new line at the beginning of each unit task, and the clock time is the time at the beginning of the unit task. *Reprinted from Card, Moran, & Newell (1983, Fig. 10.7) with the permission of Lawrence Erlbaum Associates, Inc.*

Figure 10 portrays a trace of the working memory load of the user, based on logical considerations of when information must be available in memory in order to do a task. As we can see, memory requirements build from a low point at the start of a unit task to a high within the unit task, where information for the task has been assembled, and back to a low point at the end of the unit task, where the information is no longer needed. If the information in working memory should reach a level higher than the working memory capacity, then user performance will suffer, usually manifested by the user committing errors. To avoid these errors, the user must break down the overall task into smaller (unit) units, each of which can be managed in the available working memory.

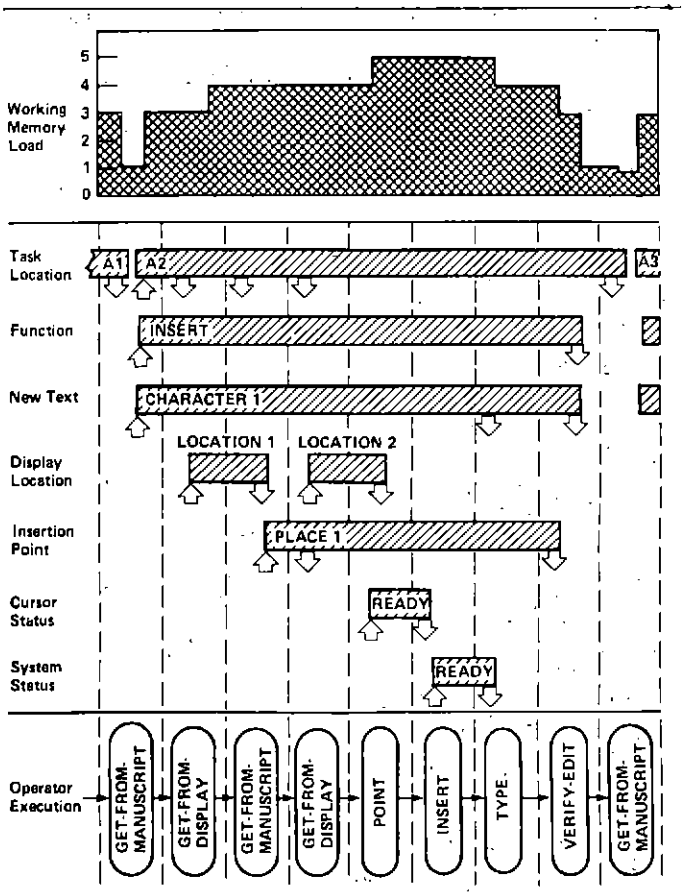


Figure 10. Data in Working Memory during a unit task. This figure is an hypothetical trace of the performance of one unit task. Time runs to the right on the horizontal axis. The bars indicate the time during which each piece of information about the task is needed. The arrows indicate the initial time data is available and the subsequent times it is needed. The histogram on the top plots the total Working Memory load over time, showing how the load peaks within a unit task and dips between unit tasks. *Reprinted from Card, Moran, & Newell (1983, Fig. 11.12) with the permission of Lawrence Erlbaum Associates, Inc.*

Unit-task behavior is at present a theoretical notion based on empirical observation (see Card, Moran, Newell, 1983, for further discussion). But it represents an important feature of user behavior—a sort of "cognitive rhythm"—that should be taken into account in designing user-computer interfaces that are sensitively tuned to the user's capabilities.

The Psychology of Human-Computer Interaction

In 1978, one of us had completed a doctoral thesis (Card, 1978) that consolidated much of the work we had been doing. The thesis helped convince us that the time was appropriate for writing a book presenting our vision of an applied science of the user. Work on the book gave us the opportunity to focus on the larger vision instead of just the pieces, and we became aware of some missing unifying theoretical work that we felt we had to do.

Although we had several calculational models (Fitts's law for the mouse, GOMS models, the Keystroke-Level Model, plus other minor models), there were questions of relating these to the classical literature in cognitive psychology and human factors. We had long sought a unifying framework for tying together the relevant psychological knowledge about users (one of our 1974 working papers called this conception the "Handbook of Cognitive Man"), but had made little progress. In preparing the book we revived this goal and came to a much more satisfying conception, the Model Human Processor, which captured the relevant psychological literature in the terms of a unified, approximative model. This model also provided a foundation for our other models.

The other big missing piece for us was to understand the relationship between the cognitive skill we had discovered and the classical notion of problem solving in cognitive psychology. Here we built a theory of the behavioral continuum between problem solving and cognitive skill and showed how practice on a task would gradually change problem solving behavior into skilled behavior. This is an active area of research in cognitive psychology today (e.g., Anderson, 1981).

The result of these efforts was the book, *The Psychology of Human Computer Interaction* (Card, Moran, & Newell, 1983). However, the book represented only the main line of our research efforts that fit together into a tightly knit view. There were many other areas of our work that we decided not to put into the book, such as the issues of learning and of users' mental models.

4. The Conceptual Interface: Mental Models

The early effort of the AIP Project was focused on understanding expert user performance. As we explained in the previous section, skilled performance is characterized by methods that users know and quickly execute to

accomplish tasks. That is, we were focused on what is typically called procedural knowledge ("how-to-do-it" knowledge). However, from the beginning, we were aware that methods are not sequences of meaningless actions, but that expert users also have an understanding of what the procedures cause the machine to do. That is, the expert users have some sort of *mental model* of what is happening inside the computer ("how-it-works" knowledge).

The first AIP memo in 1974 proposed the notion of the "user's model," which refers to the conceptual model that the user can have of the system. A user's conceptual model is distinct from (but related to) the designer's implementation model. It is an abstraction of the system's architecture and software structures—the conceptual entities that the architecture and software implements—that is simple enough for non-technical users to grasp. (For example, a user might not know how the mechanism of the delete buffer of a text-editor works, but would simply know that the deleted text goes into a "clipboard.") A user's model would typically include knowledge of where information is stored (core memory, local disk files, remote file servers). It is important for the user to have an understanding of these kinds of features, for they are often not visible to the user.

The user's model provides an integrated package of knowledge that allows the user to predict what the system will do if certain commands are executed, to predict the state of the system after the commands have been executed, to plan methods for novel tasks, and to deal with odd error situations (by characterizing the system's state according to the model, then choosing operations necessary to leave that state).

Early Encounters with Conceptual Models

Early in our research we tried to write down rudimentary descriptions of the elements of conceptual models, which included the conceptual objects, their properties, how they related to each other (the characters in a text editor, for example, including the notion of a blank space as a character), and the conceptual operations that could be performed on those objects (inserting, moving, or deleting characters, including blank spaces).

Given the obvious importance of this kind of knowledge for the user, we were surprised to find that almost no system documentation ever clearly laid out a conceptual model of the system for the user. We were also surprised to discover what a difficult inductive task it is to specify such a model, even for a seemingly simple text editor. It was a detective game in which we were forced to hypothesize and test elements of possible models in order to find a succinct conceptual characterization of how the system worked. It was a game that took us days (not minutes) to do, hence not a game in which busy users were likely to engage.

It is clear that users attempt to make sense—by building mental models—of the behavior of a system as they use it. If a simple model is not explicitly or implicitly provided, users formulate their own myths about how the system works. These user-invented models may be inaccurate or misleading outside the very limited situation from which they emerged. Therefore, we believed that if the user is to understand the system, the system has to be designed with an *explicit* conceptual model that is easy enough for the user to learn. We call this the *intended user's model,* because it is the model the designer intends the user to learn. Just what mental model the user actually forms is another issue, which depends on how clearly the intended user's model is designed, how well it is implemented, and how well it is documented.

Although we were very concerned about the mental model issue, we didn't pursue explicit studies for several reasons: we didn't have a satisfactory methodology for studying it, we didn't have satisfactory representations of it, and we were busy pursuing the performance issues we have discussed. Still, we felt that the intended user's model was an important consideration in the user-interface design process.

User Interface Design Methodology

The practical application of our concerns came when Xerox began arranging the technology transfer between PARC and the Systems Development Division (SDD), which was created to develop office system products based on the research at PARC. In the Spring of 1976, a joint PARC/SDD committee (which included one of us) was formed to advise SDD on the design of the user interface of the office systems.

The committee decided not to try to design an actual interface, but to propose a methodology for SDD designers to follow in designing their interfaces. The methodology (Irby et al., 1977) included four parts: (1) analyze what tasks the user will want to do and the steps they go through to accomplish the tasks; (2) design an intended user's model in terms of which the tasks may be cast; (3) design a command language to make that model work; and (4) design an information display to reflect the operations of the system in terms of the conceptual model.

Thus, we recommended that the designer should lay out an intended user's model before designing the command language and the information display. The whole design effort should be oriented toward keeping this model "under control," i.e., keeping it simple, consistent, and clear enough for users to grasp.

The original designers of the Xerox Star interface, the workstation product SDD built, used this methodology. The conceptual model was clearly laid out in the system's functional specifications, and the designers worked hard to keep the model consistent. Although this model was represented informally, the fact that the designers focused on it contributed heavily to the widely recognized success of Star's user interface. The conceptual model is an under-appreciated aspect of Star's interface, but Star's more

widely-touted icons and desktop metaphor only make sense with respect to its underlying conceptual model.

William Newman, another member of the committee, presented this methodology in the second edition of his book with Bob Sproull (Newman & Sproull, 1979).

Empirical Studies

By 1979, we were ready to tackle some empirical studies in order to understand the role of the user's model. The first study was an attempt to elicit the knowledge that real users of real systems have about the systems they use every day. The system we chose to study was the Alto Executive, a system which was in wide use at PARC for several years by nontechnical support people as well as programmers.

We wanted to find out what Alto users knew about the Executive. The goal was to see whether we could find some kind of mental models buried in the user's knowledge. The methodology we used was to transcribe interviews with several users into logical propositions and then classify them into categories.

The surprising result was that many nontechnical expert users (e.g., secretaries who used the system effectively every day) gave no evidence of having anything but very shallow models of how the system worked. (Perhaps we shouldn't have been surprised, because no conceptual model was documented and training was informal.) This led us to consider more carefully the role of the user's model.

In order to characterize the role a user's mental model would play in the use of a system, we performed experiments with the simplest kind of system we could devise, a simple stack-based calculator (Halasz & Moran, 1983; Halasz, 1984). We thought that a model of a stack might well be useful to help rationalize what for many users is a nonintuitive postfix command language. The formal experiment compared one group of users who were taught an explicit conceptual model of the stack, and a group of users who were carefully shielded from the stack model. The model group was trained in relation to a specific model and were told how that model related to the methods for solving arithmetic problems. The no-model group was only taught specific methods for performing the same tasks. Then we gave the two groups sets of problems that were categorized as simple routine tasks, slightly more complex tasks, and very difficult "invention tasks" (which required the user to invent new methods to solve).

The results (see Figure 11) revealed no difference between the two groups in both the routine and complex problems, but the model group performed much better on the invention tasks. The most surprising result was that even some of the no-model group were able to perform some of the invention tasks. We wanted to account for these results according to the cognitive theories we understood, so we analyzed the protocols gathered from the users as they performed the tasks.

Problem Type	No-Model Users	Model Users
Routine	98	95
Complex	87	88
Invention	25	67

Figure 11. Percentage of problems correctly done in the calculator experiments. There were two groups of users; one was taught an explicit conceptual model of the calculator's stack and the other group was prevented from having such a model. There were three types of problems, each of a different difficulty relative to what the users were taught.

Problem Type	No-Model Users				Model Users			
	Skilled Method Execution	Problem Solving			Skilled Method Execution	Problem Solving		
		Model Space	Methods Space	Task Space		Model Space	Methods Space	Task Space
Routine	90	0	8	2	89	11	0	0
Complex	94	0	6	0	91	7	0	2
Invention	2	0	84	14	7	71	19	3

Figure 12. Partitioning of the users' behavior in the calculator experiments. The users' behavior was devided into four behavioral modes: skilled method execution plus three problem solving modes, which are distinguished according to which problem space they were working in. The partitioning is based on an analysis of verbal protocol records; each line of protocol was categorized into the mode it manifested. The numbers in the table are the percentages of lines of protocol in each behavior mode. We believe that this measure underestimates the amount of work in the task space.

We divided the users' behavior into skilled method execution, as we had modeled in our earlier studies, and problem-solving. We found (see Figure 12) that in the routine and complex problems, the behavior was almost all skilled method execution: the subjects had been taught what to do, and they did it; even for the complex problems it wasn't difficult to knit together the methods they had learned for solving the problems.

Our hypothesis going into the study was that the conceptual model we taught the model group of users would provide them with a problem space through which they could search in order to find solutions. According to this theory, the user would characterize a difficult "invention" task in terms of this problem space: the state of the system when they started, the state of the system they would like to achieve, and a set of operations to move them through that space; they would solve the problem using generic problem-solving strategies.

However, when we analyzed the users protocols, we found two other kinds of problem spaces in which the users worked when they were problem solving. One was a *task space* within which they manipulated the given arithmetic task in various ways, such as dividing it into subtasks that could be solved by known methods. Another space in which they worked was a *methods space*, where they took known solution methods as strings of steps, manipulated those steps in various ways to produce new methods, and tried them to see how they worked.

We found that the most critical (although not very time consuming) problem solving was in the task space, where the user analyzed the given task into subtasks and delegated them to the model space or the methods space. The main difference between the users who had a model and those who didn't was that they had different problem spaces in which to work (see Figure 12). The model space was an effective problem space, within which the solution to the invention problems could be found; the method space was not particularly effective, but it was sufficient to allow some non-model users to stumble onto solutions to some of the invention problems (often, much to their surprise).

We concluded that mental models can be useful for novel task situations, but we found that users only use their models in specific subtasks; there was a lot of switching between the task space and other problem spaces. Users were cautious about going into a mode of behavior that involved thinking through a mental model. Model-based problem solving appears to be very mentally intensive, so users avoid it if they can apply cognitive skills. But, if users don't have appropriate methods available, they will retreat to some kind of problem solving. In these cases, a good conceptual model provides an effective problem space in which to work.

Thus, system designers should think of a conceptual model of a system as not just a simple view of a complex system, but as a problem space through which users can search for solutions to a variety of novel problems. The conceptual entities and operators in the intended user's model should be closely related to the kinds of tasks the users are likely to do, and the users should be provided with heuristics for moving through the model space.

Theoretical Studies: Task Mapping

We also worked on a theoretical analysis of conceptual models to show where they fit into the overall structure of the user interface. The Command Language Grammar formalism (Moran, 1978, 1981) shows how models relate to the task domain, the command language, and the detailed user-computer interactions. According to this theory, the conceptual model provides the user with a link between his task domain and the syntax of the interactive dialogue. That is, on the one hand the conceptual model serves as the semantics of the dialogue actions, while on the other hand it serves as a base into which the task can be mapped into the system.

Richard Young called this kind of mechanistic conceptual model a *surrogate model* (Young, 1983). Interactions on these issues with Young, who was exploring the domain of simple calculators, led to the discovery of a new kind of mental model—*task-action mappings* (Young, 1981). The properties of radically different calculator designs, such as algebraic versus stack calculators, could be best understood by an analysis of how well calculation tasks could be directly mapped into the actions available on the calculators. Surrogate models were completely bypassed in this analysis, which helps explain why people sometimes seem to get along with systems without having surrogate models of them.

In further work along this line, we have proposed a calculus, called *ETIT analysis*, for task mapping (Moran, 1983). The "fit" of a system to a task domain can be assessed by enumerating rules for reformulating system-independent task descriptions ("external tasks") into system-specific task descriptions ("internal tasks"). Rule-based system description techniques, such as this or Payne's (1984) *task-action grammar*, are beginning to provide a way to help us formalize the fuzzy notion of the *consistency* of a system (both internal consistency and consistency with respect to a task domain). Such techniques look promising as a way provide system designers with calculational techniques for predicting the learnability and "guessability" of systems.

5. The Task Interface: Pondering Ideas

Let us now turn to the role of a science of the user in the future development of the personal workstation. For us, the real challenges for user technology are now at the larger task level in which users are grappling with complex intellectual tasks. This is, of course, a return to working directly on a modern version of the original vision of augmentation workstations.

We now have workstations powerful enough to give each user a personal system equivalent to one that only a decade ago would have been shared with a hundred other users, and we are on the verge of major upgrades to even greater computing and communication power. We now have a much more developed base for user technology—not only techniques for designing user interfaces, but also models for understanding users. Thus, we are now in a better position than ever before to explore systems that can really augment human intellectual endeavors.

The challenge is to create systems that, through intimate cognitive interaction with users, aid them in structuring and manipulating their ideas. With such systems to help them, people will then be able to cope with more and more complex intellectual tasks of all sorts—authoring books and multi-media presentations, designing products and programs, composing music, analyzing experiments, the arguing points of law and policy, reasoning about scientific and social issues, and on and on.

The key to building such systems is to find ways by which a user can act on his ideas as objects, just as current text editors allow him to act on words as objects. This is difficult, since ideas are often tacit and ill-formed. A means is needed to *externalize* the ideas, to get them out of the user's head and into a form that can be organized and shaped. Two problems require solution:

(1) The user needs new ways to *represent* his ideas—to get the user's mind around the ideas, as it were. For this we can exploit advances in artificial intelligence and cognitive science.

(2) The user needs new ways to *manipulate* his ideas—to get the user's hands on the ideas. For this we can exploit advances in interactive computer graphics.

These problems are part of our current research agenda. In this endeavor we must not only build on our current base of understanding of the user, but also advance that understanding. For one thing, we have to launch studies into the nature of the complex intellectual tasks we wish to augment. This calls for a shift in our research strategy from studying users of existing systems to studying users of new systems that we ourselves build, which enables us to understand the nature of the tasks and the limits of users and systems for dealing with them. We give illustrations of this strategy from some of our own current projects.

Representation: Idea Structuring

Word processors and text editors, even powerful ones, are tailored for the final stages of writing a paper—crafting the text and graphics of the final product. Outline processors help with the previous stage of outlining. Although the latter are sometimes called "idea processors," it is clear that idea processing begins well before the stage of outlining. A genuine idea processor should allow the user to deal with ideas that are vague and ill-structured and help him gradually add structure as it is discovered.

We use the term *authoring* to refer to the larger intellectual task of gathering information, extracting and discovering ideas, structuring them, and finally composing them into a readable product. Authoring in this sense is a highly general task composed of generic subtasks, as illustrated in Figure 13. One begins by collecting sources of information; from these a set of notes (idea-sized units) are created to represent potentially relevant facts and ideas. As notes accumulate, they need to be filed in a structure suitable for retrieval, such as a topic hierarchy. But these notes also need structures that organize the ideas into meaningful, coherent themes. Such structures must be discovered, elaborated, supported by evidence in the notes and other sources. One then communicates these ideas by composing them into an interpretable product—a document, a slide presentation, or a browsable network of ideas.

The key research issue here is to help the users develop explicit mental models of idea structures, so they can see them, play with them, and evaluate them. This requires the invention of representations for *externalizing* ideas and idea structures. Cognitively, these representations serve the

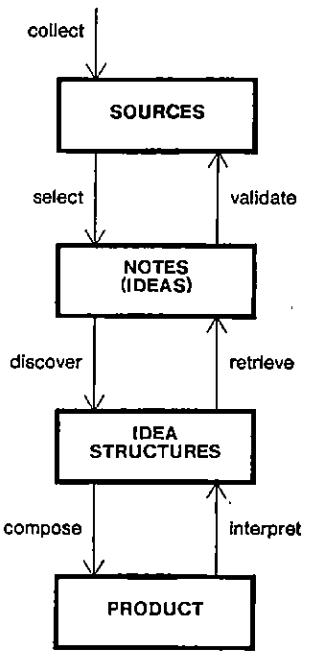


Figure 13. Schematic diagram of the generic authoring task. The boxes are the types of information to be managed in the authoring process, and the arrows are some of the processes for generating and transforming the information.

users as problem spaces; and new representations can fundamentally alter the way they are able to deal with ideas. Some representations should be "weak" in the sense that they can easily capture a large variety of even vague ideas; other representations should be "strong" in the sense that they can precisely capture and process particular classes of idea structures; and facilities must be provided to help users transform weak representations into strong ones.

We are developing a system, called *NoteCards*, to explore the broad nature of the authoring task. NoteCards supports the orienting metaphor of the notecard as a medium for capturing an idea. Notecards can be stored in fileboxes and linked into complex structures, which can then be viewed in graphic representations (such as the one shown in Figure 14). NoteCards is designed to be an open, flexible environment, so that we can use it as a "laboratory" to explore new representations and tools to support authoring tasks. Because it is an open system, NoteCards users are faced with the problem of devising appropriate ways to use the system, that is, mapping their particular tasks onto the basic elements of the system. This leads to a further interface issue of making the environment tailorable by users themselves to the wide variety of situations that we encourage them to bring to NoteCards (paper and documentation writing, scientific and legal argumentation, instruction authoring, design analysis). Thus, in contrast to our earlier methodology of observing a controlled, skilled user population, we are now observing a population of idiosyncratic, exploratory users.

Manipulation: Idea Browsing

The representations in an idea-structuring environment must be assimilated and manipulated with facility and speed. Idea-processing tasks are difficult for users, because they involve the retrieval, tracking, and manipulation of a large number of ideas (as is being attempted in Figure 14). Authoring a paper, for example, may require hundreds of notes and scores of references; programming may require hundreds of routines. But, as the Model Human Processor

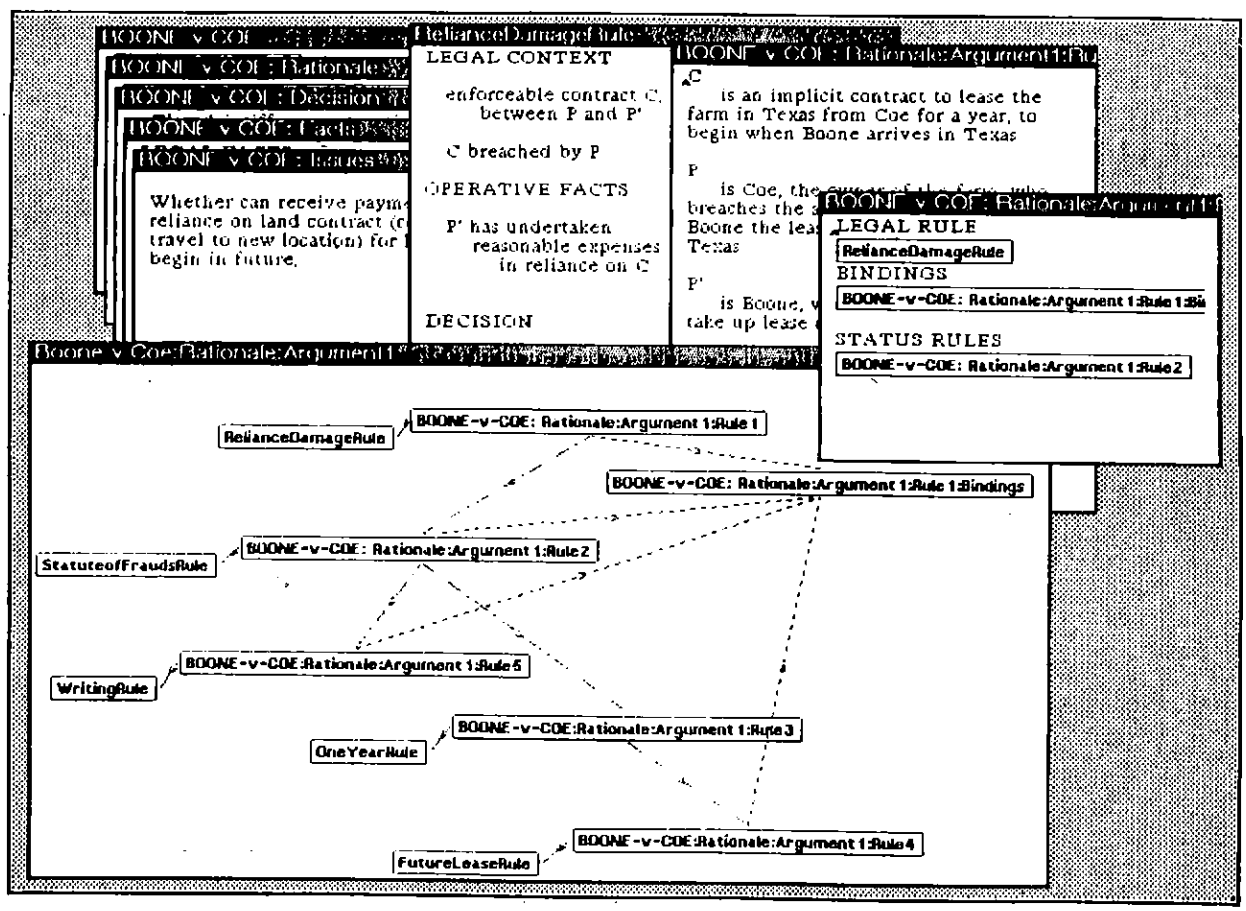


Figure 14. A screen image of the NoteCards system. The application shown is the analysis of a legal argument. A variety of legal rules must be brought to bear in arguing a particular case. The graph shows how the rules are related by applicability conditions and by bindings to the particulars of the case.

shows, only limited amounts of information can be handled in the user's working memory. These limitations lead the user to structure his behavior into unit-task bursts.

These considerations lead us to ask how a computer system can compensate for human cognitive limitations. The display can be used as an external memory to augment the user's internal working memory. This partly explains what has made the "desk-top" metaphor successful: users can use windows and icons to keep track of more documents, notes, and messages from other users than with the previous style of command language interfaces. By having these things visible as reminders, the user's effective working memory is increased, enabling him to do more complex tasks.

But the problem is that these techniques, which work well with with a few dozen objects, do not scale well up to the several hundred or even thousands of objects necessary for the idea-processing tasks we wish to do. Here, it helps to begin with a better understanding of the nature of the problem. Again, our method for gaining that understanding is through approximative models of user behavior. In this case, we have sketched a model called the Window Working Set Model (Card, Pavel, & Farrell, 1984) that analyzes access to screen objects in a manner analogous to the analysis of virtual memory operating systems. Informally, this model suggests that screen space itself is the key constraint and that at some point as the number of, say, overlapped windows required increases, user performance will decrease in a sudden and non-linear way, sending the user into the window version of thrashing.

Thus there are limitations on the uses of current graphical interface techniques for browsing large sets of ideas. Advanced graphics systems, however, open up a new set of possibilities. Figure 15 shows a fragment of an

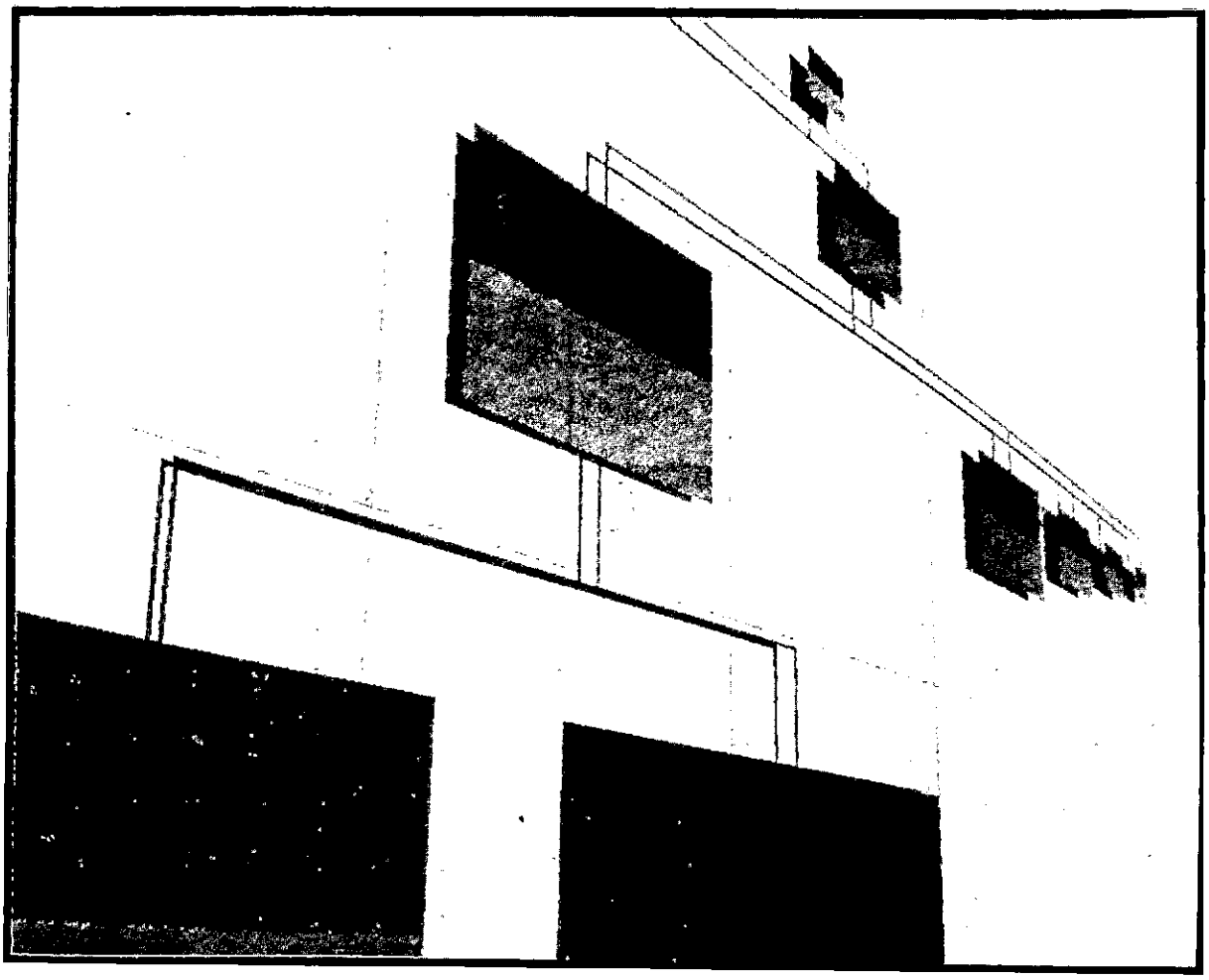


Figure 15. Experimental Dandelris browser for organization charts. The photograph shows a view of a fragment of an organization chart as seen by a user flying around this chart. The chart is laid out on a simulated 2-dimensional whiteboard. The user can change his location in simulated 3-dimensional space, his direction of gaze, and by pointing with the mouse to indicate focus, the relative sizes of different organization chart boxes. (The experimental program was written by S. Card, A. Henderson, L. Lovstrand, and B. Verner.)

organization chart from an experimental browsing system we are implementing (the DandeIris, combining a Xerox DandeTiger Lisp machine and a Silicon Graphics Iris graphics processor). The node that is the user's focus is largest and is readable in the most detail. Nodes become smaller with distance from the focus. The user can fly around this chart to see it from different points of view and change his direction of gaze; if he points at a node, indicating a new focus of attention, it grows larger. The transitions are all animated in real-time, color, three-dimensional representations to help the user keep track of the transformed identities of various objects.

Such a display is designed to help the user navigate among a large number of objects; only a few of these objects would be visible in detail at any time, but many more would be visible as orienting cues and retrieval keys. The visual movement techniques is coupled with other retrieval techniques for allowing the user to focus on a limited number of items at a time (as required by our understanding of user's processing capacity) while retaining rapid access to a large number of items (as required by our understanding of the requirements of complex intellectual tasks). By coupling experimental programming with the analysis of user behavior, we hope to find theory-based techniques to aid the user in keeping track of and maintaining effective browsing and retrieval capability for large information structures.

6. Conclusions

What has been learned about users and how does this relate to the original visions of personal workstations? The short answer is that, unlike the early 1970's when little was known scientifically about computer users, we now have a vision of the form of an applied science of the user and a few areas of knowledge where that vision has been realized. In this paper we have narrated some of our efforts at building this applied science:

At the physical interface level, we have discovered that user performance with pointing devices is constrained by the information-processing capacity of the user. We have learned the quantitative law describing this constraint and have determined that certain devices, such as the mouse, are at the performance limits allowed by this law.

At the cognitive level, we have learned that routine human-computer interaction, such as text-editing, does not involve problem solving, but rather cognitive skills based on the execution of known methods. We can see the information-processing constraints of the user show through this skilled performance in the unit task of users. We have characterized cognitive skill to the extent of developing an engineering model for use by user-interface designers.

At the conceptual level, we have learned that users often have mental models of the systems they use, and that such models enable performance of novel tasks. System conceptual models provide the basis for users to acquire mental models, and thus are an important basis for system design. But mental models are cognitively intensive, and users will avoid them by attempting to map directly from their tasks to the actions required in a system. Theories of task mapping are just beginning to emerge.

At each of the above levels, it has been possible to base an applied science on a theory of the cognitive mechanisms underlying user behavior. At each level, the applied science was shown to be practical, in particular by being influential in the design of the Xerox Star.

The challenge for us today is to use our understanding of users to discover new ways to augment users in complex intellectual endeavors. Thus, we are concentrating our efforts in those areas where users' cognitive limitations need to be overcome and where users' cognitive abilities can be aided with computer-based tools. We believe that the most interesting problems are at the task level: understanding the nature of complex intellectual tasks and finding ways to build idea-structuring tools, both representation tools for structuring ideas and display tools for browsing ideas.

Whereas we once studied users empirically using existing systems in order to understand the nature of human-computer interaction, we now use what we have learned about users to help drive the creation of new experimental systems.

REFERENCES

Anderson, J. R. (1981). *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Bell, C. G., & Newell, A. (1971). *Computer structures: readings and examples.* New York: McGraw-Hill.

Bush, V. (1945). As we may think. *The Atlantic Monthly,* August.

Card, S. K. (1978). Studies in the psychology of computer text editing systems. Unpublished doctoral dissertation, Carnegie-Mellon University, Department of Psychology.

Card, S. K., English, W. K., & Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics, 21,* 601-613.

Card, S. K., Moran, T. P., & Newell, A. (1976). *The manuscript editing task: A routine cognitive skill* (Technical Report SSL-76-8). Palo Alto, CA: Xerox Palo Alto Research Center.

Card, S. K., Moran, T. P., & Newell, A. (1980a). Computer text-editing: an information-processing analysis of a routine cognitive skill. *Cognitive Psychology, 12,* 32-74.

Card, S. K., Moran, T. P., & Newell, A. (1980b). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM 23,* 396-410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Card, S. K., Pavel, M., & Farrell, J. E.. (1984). Window-based computer dialogues. *Proceedings of IFIP Interact '84,* 355-359. London: IFIP.

Engelbart, D. (1963). A conceptual framework for the augmentation of man's intellect. In P. W. Howerton and D. C. Weeks (Eds.), *Vistas in Information Handling,* Vol. 1. Washington, D.C.: Spartan Books.

English, W. K., Engelbart, D. C., & Berman, M. A. (1967). Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics, HFE-8,* 5-15.

Halasz, F. G. (1984). *Mental models and problem solving in using a calculator.* Unpublished Ph.D. Dissertation. Stanford, CA: Stanford University.

Halasz, F. G., & Moran, T. P. (1983). Mental models and problem solving in using a calculator. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Irby, C., Bergsteinsson, L., Moran, T. P., Newman, W., & Tesler, L. (1977). *A methodology for user interface design.* Palo Alto, CA: Xerox Palo Alto Research Center.

Kay, A. (1977). Microelectronics and the personal computer. *Scientific American,* September, 230-244.

Licklider, J. C. R. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics, HFE-1* (March), 4-11.

Moran, T. P. (1978). *Introduction to the Command Language Grammar* (Technical Report SSL-78-3). Palo Alto, CA: Xerox Palo Alto Research Center.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies, 15,* 3-50.

Moran, T. P. (1983). Getting into a system: external-internal task mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Newman, W., & Sproull, R. (1979). *Principles of interactive computer graphics,* 2nd ed. New York: McGraw-Hill.

Payne, S. J. (1984). Task-action grammars. *Proceedings of Interact '84: First IFIP Conference on Human-Computer Interaction,* London. Amsterdam: Elsevier.

Young, R. M. (1981). The machine inside the machine: users' models of pocket calculators. *International Journal of Man-Machine Studies, 15,* 51-86.

Young, R. M. (1983). Surrogates and mappings: two kinds of conceptual models for interactive devices. In D. Gentner & A. L. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum Associates.

NOTES

NOTES

Conference Proceedings

ACM Conference on the History of Personal Workstations

Palo Alto, California
January 9-10, 1986

acm

ACM Conference on the

History of Personal Workstations



Conference Proceedings

Papers Presented at the Conference

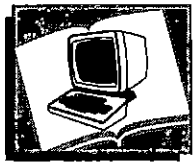
Palo Alto, California
January 9-10, 1986

Sponsored by the

Pacific Region of the
Association for Computing Machinery

ACM Conference on the

History of Personal Workstations



Conference Proceedings

Papers Presented at the Conference

Palo Alto, California
January 9-10, 1986

Sponsored by the

Pacific Region of the
Association for Computing Machinery

The Association for Computing Machinery, Inc.
11 West 42nd Street
New York, New York 10036

Copyright © 1986 by the Association for Computing Machinery, Inc., Copying without fee is permitted provided that the copies are not made or distributed for direct commercial advantage and credit to the source is give. Abstracting with credit is permitted. For other copying of articles that carry a code at the bottom of the first page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 27 Congress Street, Salem, MA 01970. For permission to republish write to: Director of Publications, Association for Computing Machinery. To copy otherwise or republish, requires a fee and/or specific permission.

ISBN 0-89791-176-8

Editor's Note

The papers contained in this proceedings are preprints distributed to conference attendees only. A final proceedings will be published by ACM as a book. Announcement of the availability of this book will be made in *Communications of the ACM*. For more information, contact: Publications Department, ACM, 11 West 42nd Street, New York, NY 10036.

John R. White
Xerox Palo Alto Research Center

Conference Committee

General Chairman and Program Chairman

Alan Perlis
Yale University

Conference Organization and Local Arrangements

John R. White
Xerox Palo Alto Research Center

Conference Support

Gloria Warner
Xerox Palo Alto Research Center

Proceedings

John R. White and Kathi Anderson
Xerox Palo Alto Research Center

Registration

Susan Mulhern
Xerox Palo Alto Research Center

Videotaping

Ken Beckman and Mark Chow
Xerox Palo Alto Research Center

ACM Headquarters Coordination

Helene S. Tannor
Association for Computing Machinery

Acknowledgements

A History conference is supposed to be a unique experience, and certainly from the viewpoint of conference organization it has been. A number of people have been involved in all phases of the Conference. The idea of bringing together researchers whose pioneering work dramatically influenced modern personal workstations originated with Bill Spencer, Manager of the Xerox Palo Alto Research Center. Alan Perlis agreed to contact potential speakers and convince them that a History of Personal Workstations Conference would be a unique opportunity -- something not to be missed. Not only did Al Perlis seem to know everyone in the field, so he knew who to invite, but he had been involved with the 1978 ACM SIGPLAN Conference on the History of Programming Languages, so he knew what perspective to convey in the invitation. The quality of the program is a credit to Al Perlis' persuasive personality.

Adele Goldberg suggested that ACM sponsor the Conference to give it an appropriate home and to help provide the infrastructure needed to pull off a meeting of this size. Throughout, Adele has been a constant source of good ideas. She suggested that the meeting be videotaped and agreed to edit the book that will represent the final conference proceedings. She has also played a leading role in using this Conference to launch a new series of ACM sponsored conferences on the History of Computing. Adele's enthusiasm and excitement have been infectious.

I want to thank a number of people at the Xerox Palo Alto Research Center for their help with this Conference. Kathi Anderson of the Computer Science Laboratory maintained constant contact with authors and helped immensely in producing the conference proceedings. Gloria Warner and Susie Mulhern provided excellent advice on local arrangements and took on full responsibility for conference registration. Videotaping a meeting of this size and editing a final set of tapes is a complex task that has been handled with cool, professional calm by Ken Beckman and Mark Chow.

Helene Tannor, SIG and Regional Conference Coordinator at ACM Headquarters, has been a tremendous help in all phases of this Conference, from negotiating an excellent hotel contract to providing advice on the conference proceedings. Not only was Helene a continual source of good ideas, but she was constantly able to point out potential pitfalls and brick walls that I had overlooked.

Finally, on behalf of everyone involved with the History of Personal Workstations Conference, I want to express our deep appreciation to the authors for taking time from current pursuits to rummage through ancient files and resurrect the facts, data and memories surrounding the early events that have led to modern personal workstations.

John R. White
Xerox Palo Alto Research Center

Foreword

Twenty five years ago computing was stationary, ponderous and centralized. Its dominant role was to serve the critical needs and purposes of organizations and the sciences. Today matters are very different. Computation is personal, ubiquitous and expansive. Power is being supplied at and to the fingertips of the individual. Computation is an amplifier of imagination and labor. The personal workstation is the gateway through which this power is supplied, harnessed, diversified, and distributed.

The transformation was not trivial, nor even inevitable. It was accomplished by people with energy, ambition and purpose, whose visions were supported by great technical expertise and insight. Their difficulties and triumphs are the stuff of which history is made. Thus it is important that we hear from them before the personal workstation becomes so packaged, so much a featureless part of our environment that its nature and value become invisible, a mere part of the overhead of daily living.

Alan Perlis
Yale University

John R. White
Xerox Palo Alto Research Center

ACM Conference on the History of Personal Workstations
January 9-10, 1986
Palo Alto, California

Thursday, January 9, 1986

Opening Session: 9:00 - 10:15

Welcome
William J. Spencer
Xerox Palo Alto Research Center

Keynote Address: Toward a History of (Personal) Workstations
Gordon Bell
Encore Computer Corp. 1

Session 2: 10:45 - 11:45

A Personal View of the Personal Work Station: Some Firsts in the Fifties
Douglas Ross
SofTech/MIT 19

Session 3: 1:00 - 3:00

Some Reflections on Early History (Abstract Only)
J. C. R. Licklider
MIT Laboratory for Computer Science 49

The Arpanet and Computer Networks
Larry Roberts
Net Express West 51

Session 4: 3:30 - 5:30

Mathematical Laboratories: A New Power for the Physical and Social Sciences
Glen J. Culler
Culler Scientific Systems Corp. 59

The Augmented Knowledge Workshop
Doug Englebart
MacDonnell Douglas 73

Banquet Presentation

The Dynabook - Past, Present, and Future
Alan Kay
Apple Computer, Inc. 85

Friday January 10, 1986

Session 5: 8:30 - 10:30

Personal Distributed Computing: The Alto and Ethernet Hardware
Chuck Thacker
Digital Equipment Corp. Systems Research Center 87

Personal Distributed Computing: The Alto and Ethernet Software
Butler Lampson
Digital Equipment Corp. Systems Research Center 101

Session 6: 11:00 - 12:00

The LINC Was Early and Small
Wesley Clark
Sutherland and Sproull 133

Session 7: 1:00 - 2:00

Hewlett-Packard and Personal Computing Systems (Abstract Only)
Chuck House
Hewlett-Packard 157

Session 8: 2:30 - 4:30

A History of the Promis Technology: An Effective Human Interface
Jan Schultz
Promis Information Systems 159

User Technology -- From Pointing to Pondering
Stuart Card and Thomas Moran
Xerox Palo Alto Research Center 183

TOWARD A HISTORY OF (PERSONAL) WORKSTATIONS (DRAFT)

C Gordon Bell
Page Farm Rd.
Lincoln, Massachusetts 01773

I originally accepted this keynote honor for five reasons: to respond to Alan Perlis' request (he told me I could present anything from a new taxonomy to personal reminiscences); second, to identify the important artifacts that should be preserved in The Computer Museum; third, to posit a framework of the history of workstations that can be written in the next century (we're all too close to create it); fourth, to summarize my own involvement on interactive computing including timesharing and large personal computers (both styles of use led to personal workstations); finally, to record my work on Digital's VAX Homogeneous Computing Environment that was totally announced last year. I led the architecture for this environment over the decade from 1973 to 1982 and was strongly influenced by the personal workstation computing movement including Ethernet. This work hasn't been described outside of product brochures.

INTRODUCTION TO THE CONFERENCE

In June 1976, an international conference was held resulting in a collection of views published in "The History of Computing in the Twentieth Century" (Metropolis, Howlett and Gian-Carlo Rota, 1980). The participants presented work up to the early 50's and thus had the benefit of 30 years of hindsight. The late mathematic's historian, Kenneth May, outlined the pitfalls of participant-written histories, commenting: "Historical description requires a time-lagged approach and means getting into understanding things as people understood them then, not as we understand them now." He also urged everyone to be open. Most autobiographies rewrite the past from the present perspective. Good diarists and notetakers have reference points. The best we can do is provide grist for the mill of the historian.

The History of Programming Languages Conference held in May 1977 (Wexelblat, 1981) provided a timeline to 1970 as context with detailed coverage of: Algol 60, APL, APT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/1, SIMULA and SNOBOL. This covered a period of initial developments from 1954 (FORTRAN) to 1967 (APL) with evolutions up to the present. Ten to twenty years of hindsight gave them the ability to select topics.

If this conference were limited to a 15-year rule, applied by the Annals of Computing History, no one would be describing personal workstations. Only a few experiments existed, but people were building systems with multiple graphics terminals and a workstation industry on a profession-by-profession basis started to form. Twenty years ago, most of us were trying to make our newly designed interactive, timesharing system work reliably and economically. In my own case, having also worked on small machines that ultimately became the minicomputer, I saw two independent threads for economical computing including interactive computing: large shared systems and dedicated small computers.

Personal workstations, like other man-made objects, appear strictly evolutionary, going through the following stages:

50's	idea (documented article, proposal, movie, etc.) stimulated through early standalone use of small computers (e.g., LGP-30, G-15)
60's	breadboards to demonstrate the idea and selected use of large computers with graphic displays (e.g., DEC PDP-1, IBM 7090, LINC, TX-2)
70's	(early) limited use of interactive shared workstations using graphic display terminals connected to minicomputers; establishment of an industry to supply terminals and professional applications software
70's	(late) working prototypes of personal workstations with concept testing through use in a complete environment (Xerox PARC)
81	introduction of personal workstations by Apollo, SUN, Xerox, and first use by early adopters
83	full-scale use for selected professionals, many companies formed, JAWS--Just Another Workstation (term) coined
85	healthy industry with evolutionary product cycle and beginning shake out of suppliers
?	steady state supply to captive users
?	decline through replacement or superposition of functions in some other form of information processing appliance (e.g., conventional Personal Computers)
?	extinction

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0001 $00.75

Hence, just as we should concentrate on tracing various workstations through these stages, we may also treat this conference about the first phases of development. More hindsight will be needed to write history

Waiting longer, which gives a future view, has the risk that extinction will come through agglomeration with conventional computing. Alan Perlis urged me to come to this conference because he feared this extinction. Conventional Personal Computers, that is the evolving IBM PC, will become the de facto workstation for virtually all applications by 2005. The latest chip introduction by Intel strongly permits and supports this to occur, if history is any indicator.

The real workstation phenomena and industry is not well described or recorded at all because most of the products have come from industry where individuals are not rewarded for writing papers but producing and selling machines. For example, Doug Drane and the ATEX system that produced law and typesetting offices, doing essentially what a STAR does, only five earlier and less expensively; GE's genigraphic system for the graphic artist; Bill Poduska and the Apollo workstations; or Applicon, Calma, Computervision, DEC, Evans and Sutherland, IBM, REDAC, Wang, Datapoint's ARCnet, [illegible] No university (other than Utah and Brown) really participated until 1980. Noticeably absent are Dave Evans, whose direct contributions or indirect contributions through products at Evans and Sutherland, and graduate students who went on to create much of the science behind workstations. Ivan Sutherland's work was recorded. Xerox PARC was a noteable exception, which published its work in a delayed fashion and was marginally successful at taking the research into products in a commercially viable fashion.

We now need to focus and agree on some definitions and dimensions, making sure the various threads of personal workstation development are explored so the historical facts can be recorded properly and eventually interpreted by real historians. We need to find the first use of the name. For example, the Second Edition of the Computer Encyclopedia, 1983, uses the word three times, but in a somewhat different context. By the summer of 1983, the phrase JAWS (Just Another Workstation company) could be heard in the venture capital community when they realized they had funded too many companies to build personal workstations.

WHAT IS A (PERSONAL) WORKSTATION?

Terminals connected to large computers (supercomputers, mainframes and minicomputers) and utilizing interactive professional-applications programs are the clearest antecedents. These had, as direct descendants, terminals connected to shared, but dedicated, minicomputer systems in the mid 70's to deliver cost-effective computing. An overview of the classic machines and the classes they formed is given in Figure 1.

Figure 2 shows several paths leading to the personal workstation. Large machines (with large memories) provide the major impetus. Large shared systems allow the price per terminal to fall in

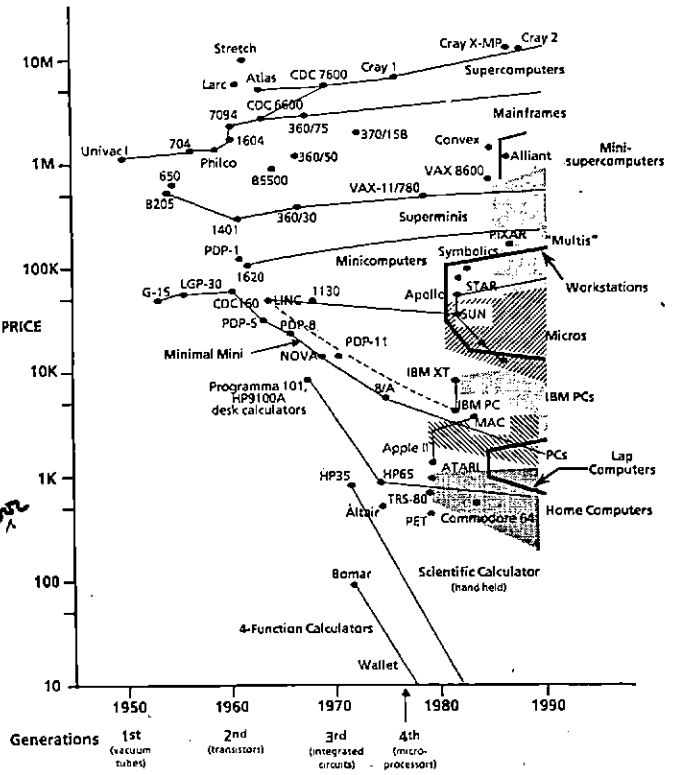


Figure 1. Computer system prices at introduction and classes versus time.

line with the salaries of the professional using the station. Four lines, corrected for inflation, depict the LINC price, the cost of a supported professional (e.g., engineer or scientist), the starting professional's salary, and a clerk's salary. The price of facilities such as a computer are "pegged" to these constants. The justification of equipment is usually based either on productivity or achieving some new level of

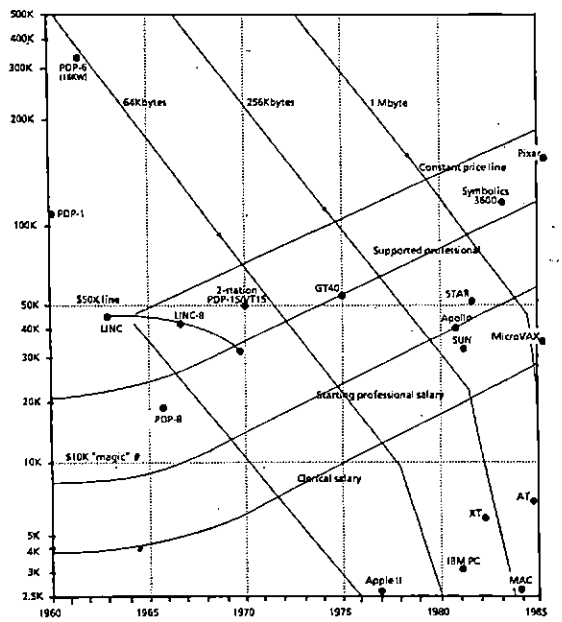


Figure 2. Workstation price versus time for various workstation introductions.

performance or capability. While salaries of clerks are greater than $10K, the workstation industry believes $10K to be a magic, highly elastic price barrier which will increase sales. In the later part of the 80's, large personal computers will easily have enough memory and capability to take over all functions provided by nearly any of today's personal computers or personal workstations. Constant price, very powerful, large workstations will continue to be built, but the mass market will be served by the rapid price decline inherent in the IBM PC evolution.

A personal workstation is a relatively large (>50 pounds) and expensive ($10,000 to over $100,000 in 1985) personal computer, with the appropriate transducers, used by a professional to carry out generic (e.g., calculation, mail and communication) and profession-related activities such as music composition, financial modeling, or computer-aided design of integrated circuits.

Personal workstations are necessarily distributed with the person and interconnected to one another forming a single, shared (work and files) but distributed computing environment... the workstation environment. A workstation's location is either with an individual on a dedicated basis or in an area shared by several members of a group. This choice is dictated by the cost and size of the workstation, relative to the cost and value of the work.

Personal workstations appeared by name about 1981 with the concurrent appearance of:

- microprocessors with at least one-half megabyte of physical memory virtual memory addressing
- memory chips of at least 64K bits providing primary memories of 1/4 to 1 megagbyte, permitting the use of large programs and construction of high resolution bit-mapped terminals
- disks of >10 megabytes
- local area networks for interconnecting the stations

Future descendants are likely to evolve from extended, lower-priced personal computers, interconnected via LANs. They would include appropriate transducers and corresponding professional applications programs.

A profession is any vocation, occupation or business, associated with work, including: secretarial, commercial, science, engineering, mathematics, and the arts.

A personal computer or pc is a self-contained computer with secondary file memory, and appropriate transducers to interface with people. A personal computer is used interactively by one person at a time, at a location convenient to the user, and may "belong" either to the person or a group. A personal computer, for a given use, is self-contained, i.e., requiring no external program or data preparation units, permitting a user to go through various stages without external intervention.

The microprocessor, memory, and mass-storage technology appearing in 1975 lead directly to the personal computer industry. Early computers utilized the simple, single process, stand-alone operating systems developed for both interactive, timeshared computers and stand-alone minicomputers. Nevertheless the first personal computer, the LINC (Clark and Molnar, 1964) was built in 1962 long before its predicted technological time.

Personal computers can be used either in a

- personal or private environment encouraging separation of files, resources and work, with personal security, whereby the only communication with other computers is via secondary memory (floppy) sharing, or transmission of messages via standard communications lines, or
- shared, workstation environment encouraging communication among the pc's, sharing of resources (e.g., files, printers) and working on a single large work assignment or goal.

An IBM Personal Computer or PC is a particular personal computer utilizing the Intel 8086...80386 architectures and the evolving MS DOS operating systems. A PC may be extended and used in a workstation environment when interconnected via LANs, and permitting access and sharing of facilities and work. Hence, constant-cost PC's are likely to evolve into the de facto personal workstations.

BELL MODEL OF MEMORY PRICE DECLINE, FORMING NEW COMPUTER CLASSES

Hardware technology improvements specifically in silicon and magnetic storage have been the sole enabling determinants of progress in computing... they are the "technological devils" that drive the formation of our industries. This is the economic basis that forms all computing classes, including all forms of workstations. Since personal workstations evolve from other computer classes, the entire hierarchy of computers must be understood. In effect, we get no product before its (technological) time!

In 1975, I observed that memory price was falling at a rate of 20% per year. Since it was a constant fraction of system prices, they also declined at the same rate. This 20% decline was based on the learning curves for manufacturing core memory manufacture and distribution. Thus, a computer would drop a factor of 10 in price, per decade.

In 1972, with the introduction of semiconductor memories by Intel, price began to decline a factor of four every three years, the cycle for new memory chip introductions. (The Four Phase company produced the first 1K MOS memory chips before 1970.) Table 1 shows the introduction of semiconductor memories and other devices, and the development of specific personal computers and personal workstations. The reduction in the price per bit of semiconductor memories amounts to a 36% price decline each year or a factor of 10 every 5 years, twice as fast as the original core memory (and discrete logic) based model.

In 1975, I also observed that memory size is the determinant of computer use and hence computer classes. That is computer power is proportional to memory size, or memory size squared when a proportionally faster processor is used. Looking at the declining prices for various sized systems, one

Table 1 - Memory Chip and Microprocessor Introductions
With Resulting Personal Computer and Workstations

Year	Memory	Micro	Width	Memory	Examples
1972	1K	4004	4		
1973		8008	8		
1974		8080/6800	8		
1975	4K	6502	8	4K-8K	Altair, IBM 5100
1976		Z-80	8		
1977					Apple II, TRS-80, PET
1978	16K	8086	16	16K-32K	
1979		68,000	16	16K-64K	Atari
1980		8088	8		
1981	64K			64K-256K	PC, Apollo, SUN I
1982		186/286	16		Commodore 64
1983					LISA
1984	256K	68020	32	512K-1M	MAC, AT
1985		386	32		SUN 3
1986					?
1987	1M		32	4M	?

can see each computer class taking on the attributes and power of its higher order neighbor. Table 2 classifies what I then thought were the capabilities of various computers versus their memory sizes. I felt then, and still feel, that a one-megabyte address space for a process is the minimum size, and OK at least in terms of what we actually use. Anything smaller (e.g., 64Kbytes) is shear hell to use!

My goal in 1975 was to be able to use the past to project the evolution of simple computer classes to 1980 as the basis for the VAX evolution. Figure 3 shows the price of various systems versus their memory size. Note that a new computer class emerges when the price drops an order of magnitude. This evolutionary model was also described for DEC's hardware development (Bell, Mudge, and McNamara, 1978) shown in Figure 4. This evolutionary model turned out to be quite accurate, predicting both the Apple II (just 3 years away), low-cost shared micros, and the new class of mini-supercomputers that are beginning to emerge this year.

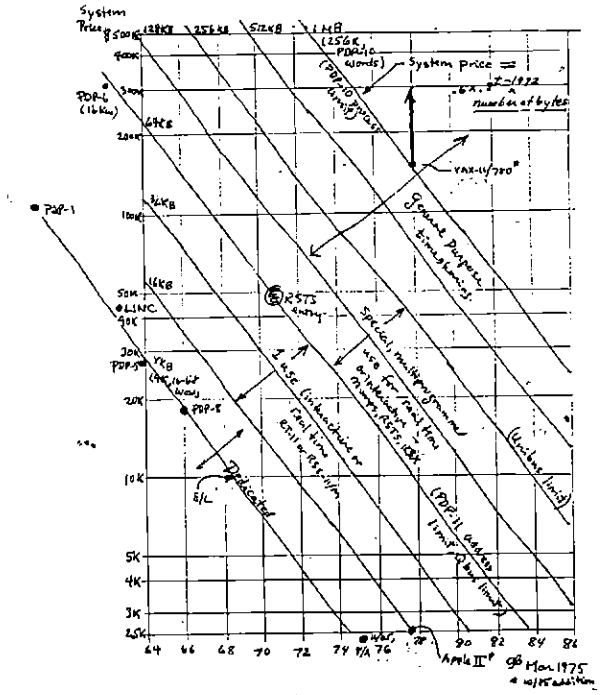


Figure 3. System price versus time for various memory sizes and system types (Bell 1975).

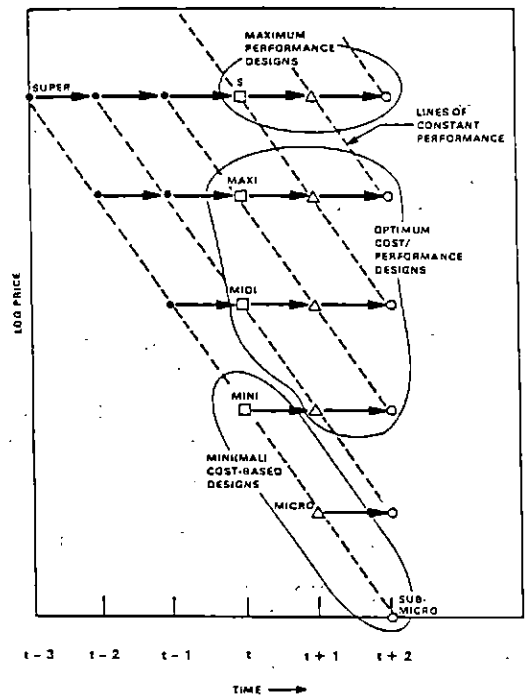


Figure 4. Price versus time for each machine class.

Since computer price determines the performance and therefore the economics and style of use, I began to observe at about the same time, that there were roughly three levels of computer use: central (mainframe), departmental (minicomputer), and personal. Dave Nelson, formerly at DEC, Prime and now at Apollo, extended this view to several other computer classes and introduced a model for price and weight (Table 3). These emerging smaller classes put significant pressure on interconnecting all the computers and forced the need for Local Area Networks (LANs). Figure 5 shows the style of use and environment for various computer classes. Note that personal workstations provide roughly the same capability of larger shared computers, and are large personal computers interconnected via LANs. IBM appears to have recently adopted this hierarchal view of computing and assigning meaningful work to all the levels, so it must be right (at least for a few years).

Table 2 - System Structure, Memory Size, and Resultant Use (G.Bell 1975)

Structure	Memory Range	Function (use)	
Dedicated (fixed-1 use)	4KB - 8KB	Interactive—e.g. POS cash register Real time—e.g. scope, traffic control, automobile	Special purpose, Fixed
Programmable (1 user)	16KB - 65KB	Interactive—RT11 (CP/M) Real time—RSX11S, M	Small scale, generality
Dedicated (multi-programmed) n-users)	65KB - 256KB	Interactive—MUMPS, RSTS, Trans.Proc. Real time—RSX-11M, D	Special purpose
Programmable (multi-programmed n-users)	128KB-1024KB	Interactive—IAS, TOPS 10, RSTS Real time—RSX-11D	Generality

Table 3 - Nelson+/Bell Computer Classes

Tier	Location	Names	Price($)	Weight(#'s)
0	wallet	calculator	10	.05
1	pocket	calculator, special function unit	100	.5
2	briefcase	kneetop, small personal	1K	5
3*	office	personal, large personal,"workstation"	10K	50
4*	department	shared micro, minicomputer, super-minicomputer, mini-supercomputer	100K	500
5*	center	mainframe	1M	5,000
6*	region	supercomputer	10M	50,000**

+ Founder and Vice President, Research, Apollo Computer Corporation
* Based on Bell's Distributed Processing Computing Model (circa 1975)
** Perhaps, if heat exchanger is included

NETWORKS AND THE SHARED AND PERSONAL WORKSTATIONS EVOLUTION

Although technology is a major variable that creates computer classes, the development of networks and changing patterns of use also influenced the evolution of personal workstations. The impact of the network and local area network on workstations is based on my perspective as chief architect of the ensemble of DEC products until 1983.

DECNET AND THE INFLUENCE OF ARPANET

The various DECNET architectures were initially created in 1974 by Stu Wecker while a member of DEC's research group. They were based on the notion of hierarchical computing according to use, concurrent with Digital's work on distributed processing. The imperative was to interconnect the hierarchy of DEC's minis and mainframes as tightly as possible forming a single computing environment.

The need for DECNET occured because Digital was proliferating timeshared and real-time minicomputers that had to have a way to communicate with one another and with its own and other central mainframes. DECNET was introduced in the following phases:

I	75	task-task, file transfer, file access; point-to-point; 8-bit net addreas; operating systems: RSX-11 D, M, IAS
II	77	improved system generation; used for first DEC engineering network (now several thousand machines at tens of sites; operating systems: rt-11, rsts
III	80	routing, virtual terminal, full network management; operating systems: decstystem 20, vms
IV	83	large routing address, full remote rms files; c-terms; ethernet support
	85	portable DECNET (by Wecker's company)

The basic model for DECNET was ARPANET (Roberts, 1970), with the important difference that a network need not have specific IMPs or TIPs to do the packet switching for the worker computers. That is, the worker computers could do the packet switching for the network, and when the packet switching load became too high, one inserted packet-switching computers.

XEROX PARC ETHERNET/ALTO ENVIRONMENT, CARNEGIE-MELLON SPICE PROPOSAL

In 1979, Carnegie-Mellon published a proposal solliciting vendors for a personal workstation environment. Their statement, which had the effect of stimulating the design of various workstations was:

> "The era of time-sharing is ended. Time-sharing evolved as a way to provide users with the power of a large interactive computer system at a time when such systems were too expensive to dedicate to a single individual... Recent advances in hardware open up new possibilities... high resolution color graphics, 1 mip, 16 Kword microprogrammed memory, 1 megabyte primary memory, 100 megabyte secondary memory, special transducers,... We would expect that by the mid-1980's such systems could be priced around $10,000."

Although the concept was correct, the details of the machine were wrong because evolving technology was not considered. This led to selecting the wrong vendor, the late Three Rivers or PERQ Systems. Three Rivers' founder, Brian Rosen, a CMU grad who had worked at Xerox Palo Alto Research Center (PARC), designed PERQ as his version of the ALTO and D-series machines. Carnegie-Mellon may have taken on a leadership role in personal workstation environments by building its SPICE (Scientific Personal Integrated Computing Environment) on the PERQ and then contracting to build a similar environment for IBM, provided the SPICE derivative product is marketed.

The prototype for SPICE, and virtually all other distributed personal workstations environments (e.g. MIT's Chaosnet and the personal LISP workstations that were the basis of LISP Machines Incorporated and Symbolics) was PARC's first 3-megabit-per-second Ethernet interconnecting Altos, operational as an environment in 1975 (Perry and Wallich, 1985; Pake, 1985). Datapoint's ARCnet with personal computers using the Intel 8080 was clearly the first commercial workstation environment. The Alto

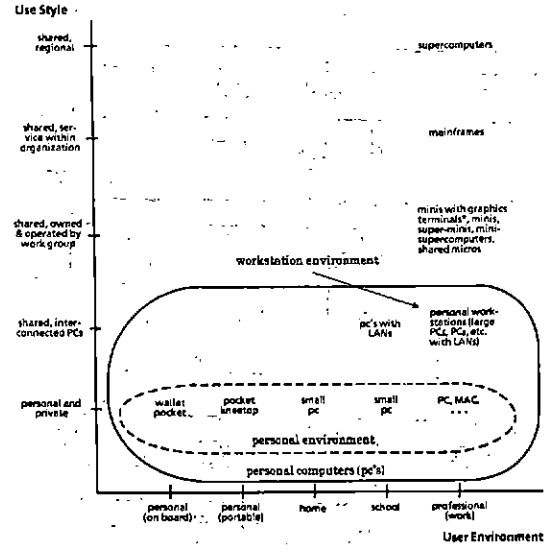


Figure 5. Computer use style versus user environment for various computer classes.

processor was a near derivative to the Data General Nova with an integrated bit-mapped display. It is important to note the PARC environment also included a timeshared computer based on the PDP-10 architecture, MAXC, which could be used for shared files and significant computation. In April 1981, XEROX introduced a product version of Alto, the STAR workstation and various file and print servers utilizing a 10 megabit per second Ethernet, but without a real central computer. The PARC Ethernet was the forerunner of today's 10-megabit-per-second IEEE 802.3 standard (as originally drafted by Digital, Intel and Xerox).

THE DIGITAL VAX, HOMOGENEOUS COMPUTING ENVIRONMENT

The idea of a complete, distributed computing environment, based primarily on the VAX architecture (but including the PDP-11 and DECsystem 10/20 computers) was presented to DEC's Board of Directors in December 1978 and approved as the strategy for guiding future product development.

The goals and constraints of this strategy is outlined in Appendix 1. The impetus was based on the early success of VAX, including plans to make a wide range of products to fill each of the 3 classes from MOS micros for personal computers to clusters of high-performance ECL VAXen. Furthermore, it was desirable to limit future evolution of the 10/20 and 11 lines, since both were providing essentially the same capability. The need for interconnection and the availability of LANs were critical as described above. Finally, computing was evolving from a centralized computing style to a more distributed computing environment as demonstrated by the Alto environment (but including high performance and departmental level computers).

The environment, shown in Figure 6, provided a wide range and hierarchy of compatible computers and ways of interconnecting them to provide users with generic (eg., word processing, mail) program

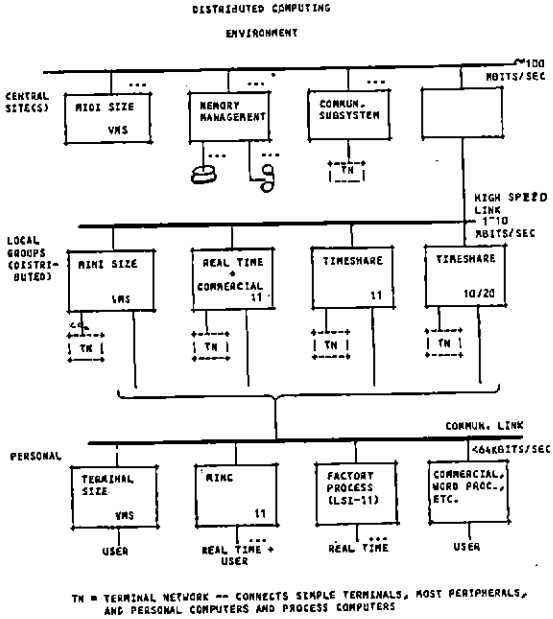


Figure 6. Digital distributed computing environment (Bell, 12-78).

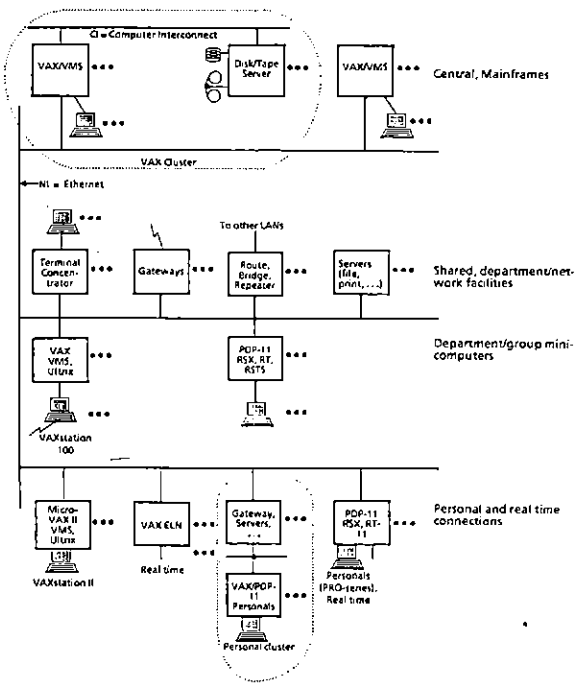


Figure 7. Digital homogeneous computing environment 1984.

development, and profession-specific computing using timeshared minicomputers or personal workstations. The goal was to provide the widest range of choices by having complete compatibility for where and how computing was to be performed without having to make a priori commitments either statically (purchase or installation) or even run-time to a particular computer system class (i.e. mainframe, minicomputer, team, or personal workstation). By 1981, the original structure evolved to require the Local Area Network (Ethernet) for connecting all computers, initially called NI for Network Interconnect, as part of the hierarchy of interconnects (CI=Computer Interconnect, NI=Network Interconnect, BI=Backplane Interconnect, and II=Interchip Interconnect; II did not materialize) as shown in Figure 7.

The design of the environment is substantially more than the design of a single range of compatible computers because diffent styles of user are required depending on the machine class, and all the computers must be interconnected and work together in a multi-level hierarchy.

ETHERNET

While DEC had prototype LANs and proposals for LANs internally, I felt it was essential to have a standard. We hired Bob Metcalfe as a consultant and his missionary role was to make Ethernet the atandard using Intel's chip design capacity and Xerox's patent and LAN experience. The IEEE got involved once they heard the idea of the LAN. They went on to facilitate the design of many other "standards" which, of course, had the effect of diminishing the notion of a standard. Fortunately, the development of Ethernet crept into existence slightly more rapidly than the IBM Token Ring LAN which was introduced in October 1985, otherwise I doubt if there would be any significant use of LANs today. I understand about 30,000 Ethernets exist today.

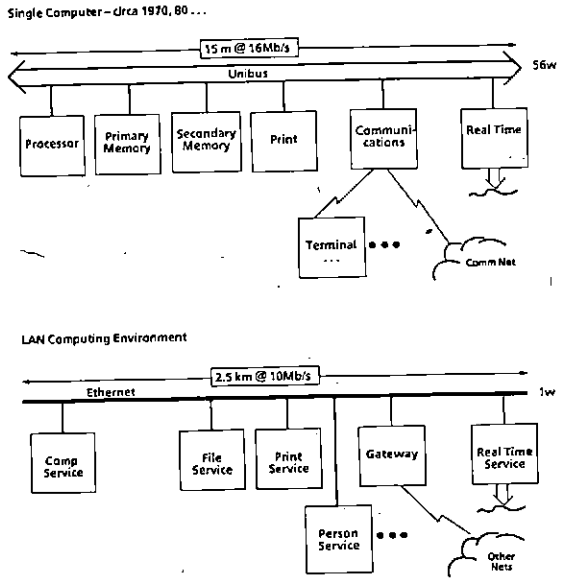


Figure 8. Unibus (c. 1970) and Ethernet (c. 1982) computer structures comparison.

In January, 1981 when Ethernet was being introduced by various vendors and being completed as an international standard, several of us from the developer companies (myself from Digital, Noyce from Intel, and Liddle from Xerox) made a broad appeal to the U.S. and European press and manufacturers on the importance of the standard. My theme was: "Ethernet is the Unibus of the Fifth Generation." We all argued that the standard was essential:

- to interconnect department-level minicomputers and mainframe computers to one another, permitting file-to-file transfers, remote-terminal emulation, and distributed processing among the shared computers;
- as a switch for interconnecting terminals to the higher level computers within the environment;
- to centralize the gateway function in one place, versus requiring every computer to have protocol conversions to communicate outside the Ethernet environment;
- for building fully distributed computing environments. The PARC model of a computing system was to decompose computing into a series of functional servers such as printing, filing, and with individual workstations. The analogy with Unibus (circa 1970) as a computer versus Ethernet (circa 1981) as a computing environment is clear when comparing the two structures (see Figure 8).
- as a bus for building computer systems; and

THE ENCORE CONTINUUM

With the formation of Encore Computer Corporation in 1983, our goal was to provide a complete environment, The Continuum (Figure 9), like the VAX environment, but based on a single microprocessor (Bell, 1985a). The "Continuum" provided both the distributed personal workstation style (described in the evolution, Appendix 1) and shared central computing using a multiple, microprocessor architecture, which I have named the Multi (Bell, 1985b). Ethernet provides the switch for interconnecting terminals to the Multimax using concentrators which pre-process terminal requests for communicating with other environments and for distributed-workstation environments. There are no terminal connections to the shared computers (something I have been trying to eliminate for years).

APOLLO DOMAIN

Dave Nelson, founder of Apollo and Domain architect (May 1980), was in the research group at DEC during the formation of DECnet and VAX. In the late 70's he went to Prime to work on the Prime token local area network and then left to develop the Apollo Domain architecture. Domain, introduced in March 1981, most typifies today's personal workstation because it provides a single addressing scheme for accessing all the memory (files) of a system. Dave credits me with stimulating this unifying architecture which is like the Unibus in terms of addressing.

WORKSTATION EVENTS, DIMENSIONS AND TIMELINE EVOLUTION

This section serves as a model for future historical research because it outlines, albeit incompletely, some critical events in workstation evolution.

The evolution of the physical workstation hardware is almost indistinguishable from the computer, and hence the evolution of personal workstations parallels and is directly coupled to the development of mainframes, minicomputers, and microprocessors. The principle difference is that workstations include a significant set of transducers to interface with humans and communication links, and additional user interface (software) advances.

PROCESSOR AND MEMORY COMPUTING ENVIRONMENT

The reinvention of the wheel and features within each of the computer classes has been nicely depicted by Burger, Cavin, Holton and Sumney (1984) as a timeline of the critical processor features (see Figure 10). The single most critical dimension of processing power is virtual memory size. The reinvention of the wheel of virtual memory size is shown in the Figure 11 from Siewiorek, Bell and Newell (1982) as each architecture reinvents and evolves the concepts of virtual memory.

VIDEO ENVIRONMENT

The evolution of displays has occurred along a set of dimensions which are relatively closely correlated with evolutionary time as follows:

- performance as measured in ability to display objects, including the ability to transform the objects;
- hardware generation of the picture process. Myers and Sutherland's Wheel of Reincarnation shows the evolution of how pictures are controlled (Figure 12 reproduced from Bell, Mudge and McNamara, 1978.) Note the evolution: direct control of the display object by the central processor; display list issued to a DMA controller or external controller; separate processor with loops, subroutines; arithmetic instructions added to aid dynamic displaying; use of virtual memory (beginning to occur to manage large pictures; and removing the display processor to form a separate computer.

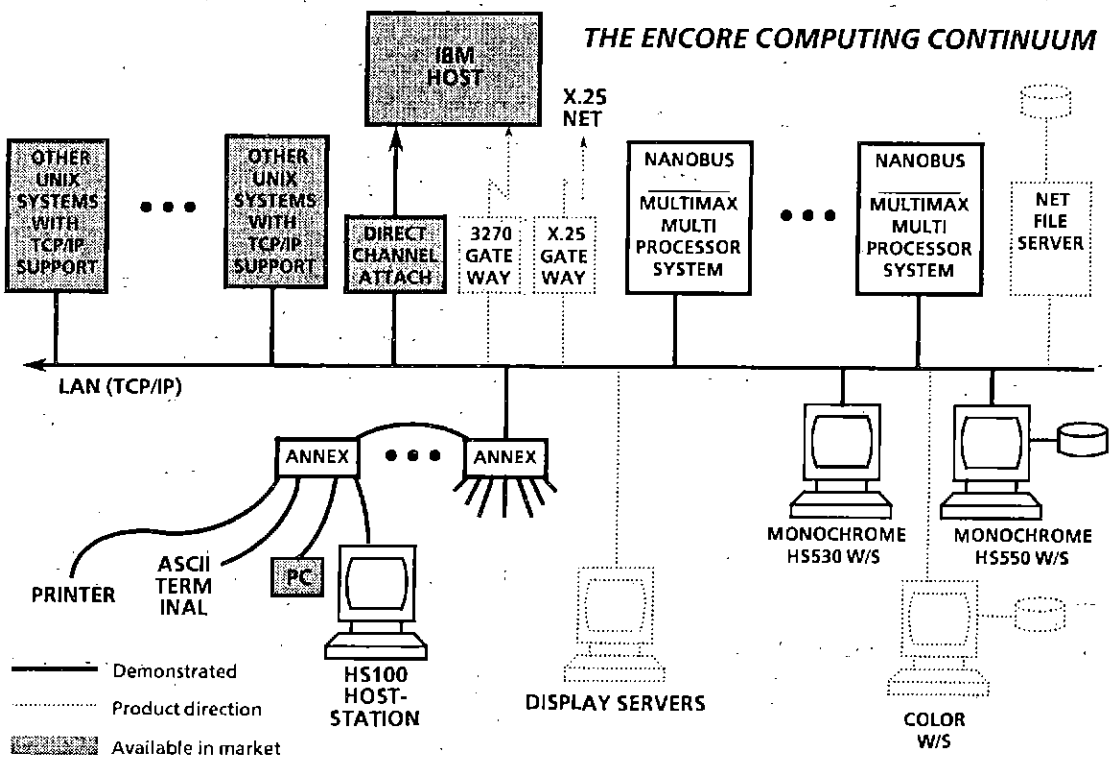


Figure 9. The Encore Computing Continuum.

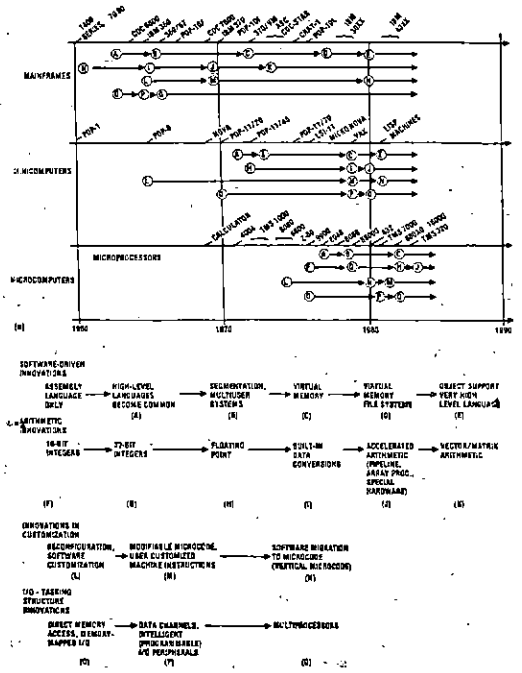


Figure 10. Computer architectural innovations for mainframes, minicomputers, and microcomputers from 1960 to 1990 (a) and the descriptive legend (b). (Taken with permission from Burger, Cavin, Holton, and Sumney, 1984.)

- resolution (in pixels or analog vector resolution);
- number of bits per pixel including use for monochrome/color;
- display data types
  - points, characters, lines, curves,
  - 3D line drawing
  - wireframe meshes, 2D surfaces, 3D surfaces,
  - shading, sculptured surfaces, constructive geometry
- translation of data types
  - fixed
  - translation, scaling, rotation,
  - 2D clipping, 3D perspective, 3D clipping
- standard interface evolution: direct control, Siggraph CORE Library, VDI, CGI, NAPLPS, etc;
- windowing of output images;
- character, line, picture, and 3D image input.

TACTILE AND MANUAL POSITION OR LINE INPUT

This category includes: keyboard, 2D and 3D light pens, direct touch screen, tablet, digitizing tablet, knob, joystick, track ball, mouse, etc.

PAPER OUTPUT

This section must include the evolution of: conventional printing and plotting along with early film recorders to capture images, and extending to modern dot matrix printers of all kinds for monochrome and color.

VOICE I/O

While today's personal workstations do not permit substantial voice communication, voice input such as the Kurzweil voicewriter promises to add a significant dimension for human communication. The critical voice "firsts" for use with personal workstations are relatively sparse at this time.

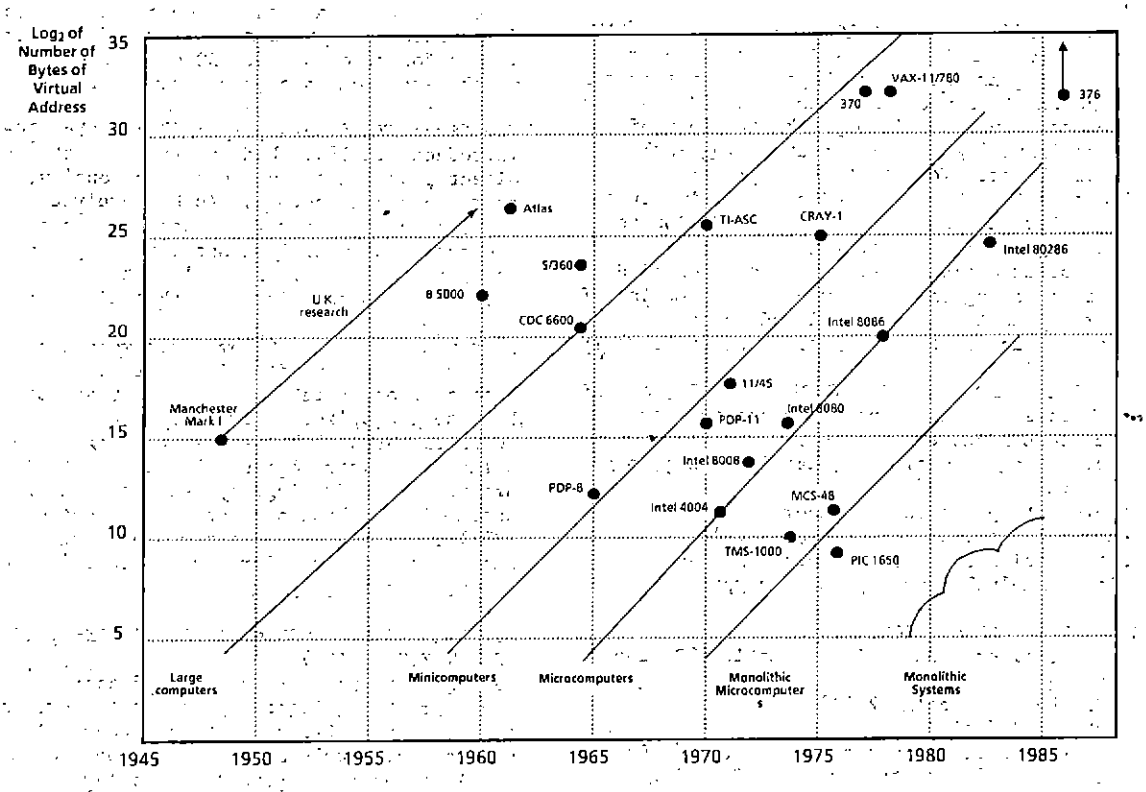


Figure 11. Virtual address space evolution for various computers and computer classes.

LINKS, SWITCHES AND NETWORKS

Links, switches and communications networks have influenced the formation of computer networks and computing environments. The previous section described their influence on the formation of the DEC computing environment. Others should be traced.

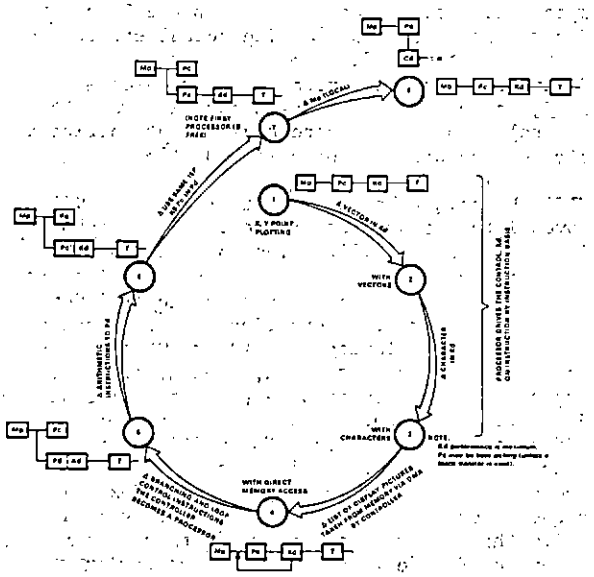


Figure 12. Myer and Sutherland wheel of reincarnation for display processor evolution.

SPECIAL PURPOSE I/O AND ROBOTS

The ability to handle arbitrary signals from other environments is crucial to the formation of various profession-specific environments (e.g., music, scientific experiments).

OPERATING SYSTEM, DATA-BASE, LANGUAGE, AND GENERIC ENVIRONMENTS

Traditional computer science history has laid claim on operating systems, databases, and languages, including their application to personal workstations. For example, the History of Programming Languages conference constructed an excellent timeline of the various languages including those for specialized environments such as interactive computing and specific application domains such as statistics, science and engineering.

Generic environments which users of traditional, timesharing computers have required include:

- TIMESHARING ORIENTED WITH TEXT AND GRAPHICS TERMINALS (SLOW LINKS)
  - text editing, mail and network mail
  - drawing and graph plotting
  - shared word processing
  - database access
  - transaction processing
- SHARED WORKSTATION AND PERSONAL WORKSTATION ORIENTED (FAST LINKS)
  - multiprocess control including control, windows
  - spread sheets
  - drawing and painting

PROFESSION SPECIFIC WORKSTATIONS

In addition to generic applications programs (eg. word processing and mail), each profession requires an environment to support and "understand" the work of the professional. This may take the form of special hardware (eg. high-resolution mega-color scopes for the graphic artist, high precision analog output for the musician, or fast simulation processors for the VLSI designer) in addition to specific programs that understand and have knowledge or expertise of the work being carried out. The following taxonomy provides an outline of the environments that have developed quite independently and that historians should research.

- SCIENTIFIC, MEDICAL AND MATHEMATICS WORKSTATIONS
  - life science with transducers appropriate to the measurement task
  - medical (eg. radiologist, cardiologist, bone surgeon)
  - geologist, petroleum engineer
  - physics experiment control and analysis
  - statistician and mathematician
  - signal analysis, encryption/decryption
- ENGINEERING WORKSTATIONS
  - electrical or electronic schematic (2D line drawings)
  - VLSI physical design 2-1/2 D
  - mechanical drafting
  - design of sheet metals or piping (2-1/2 D)
  - architectural and structural engineering
  - general mechanical design of arbitrary surfaces for analysis and fabrication
- SOFTWARE AND KNOWLEDGE ENGINEERING WORKSTATIONS
- ARTS (TYPESETTING, GRAPHICS ARTS, MUSIC, etc.) WORKSTATIONS
  - publisher of: newspaper, magazine, book
  - scholars including: historians, linguists, literary scholars
  - graphic artist, illustrator, cartoonist, choreographer
  - composer
  - artistic film production
- COMMERCIAL WORKSTATIONS
  - general clerical and transaction processing, office word processing, executive
  - financial personal workstations for banking, insurance broker
- SPECIAL CONTROL-ORIENTED WORKSTATIONS
  - air traffic control
  - process and plant control
- TRAINING WORKSTATIONS (and are these workstations?)
  - simulation of industrial plants, aircraft, etc.
  - Computer Aided Instruction such as Plato

SOME DISPLAYS, PERSONAL COMPUTERS AND WORKSTATIONS I HAVE KNOWN

This section describes various displays connected to computers, which were used as either personal computers or workstations. The MIT and Lincoln Laboratory displays embedded in Whirlwind, TX-0, TX-2, and LINC were among the earliest personal computers. Since these computers (except LINC) were one of a kind, their impact was as prototypes for subsequent developments. One direct impact was in the formation of the DEC computers, many of which were used as personal computers.

The first DEC display, Type 30 for the PDP-1 introduced in 1961, had instructions that permitted direct plotting of points and optionally, characters and vectors. As DEC evolved and built timesharing computers with operating systems which protected the user from accessing input/output equipment directly, the ability to have highly interactive terminals and workstations decreased.

Nearly all of the DEC computers were used as components for building interactive, "workstation-style" systems. DEC and Original Equipment Suppliers (OEMs) built and sold "workstation-style" systems for use in the sciences, printed circuit layout, mechanical and architectural drawing, typesetting and office automation. These pioneering applications (circa 1970) operating on dedicated timesharing systems served as the basis for much of today's evolving workstations.

MIT WHIRLWIND, LINCOLN LABORATORY TX-0, TX-2, AND IBM AN/FSQ7 (SAGE)

During Whirlwind's first year of operation, 1950, Bob Everett published a paper that showed a picture of its display (with an attached camera) and listed some actual problems carried out on the machine.

1. An industrial production problem for the Harvard Economic's School
2. Magnetic flux study for Whirlwind's magnetic storage work
3. Oil reservoir depletion studies
4. Ultra-high frequency television channel allocation for Dumont
5. Optical constants of thin metal films
6. Computation of autocorrelation coefficients
7. Tape generation for a digitally-controlled milling machine

Whirlwind was initially established as a large specialized simulator to the Airplane Stability and Control Analyzer (ASCA). The actual use turned out to be a prototype for the SAGE (Semi Automatic Ground Environment) system for air defense and SAGE, in turn, was the basis for air traffice control. The first successful use of Whirlwind for air defense occurred on April 20, 1951.

MIT's Lincoln Laboratory, established in 1951, moved into its quarters in Bedford, Massachusetts, in the summer of 1953. Lincoln built the TX-0 computer to test the use of transistor circuitry and a large core memory and then the TX-2 for large-scale computing experiments. Both had a point-plotting display with 10" x 10" area and 1K point resolution, light pen, camera, switch input, and abilities to interconnect arbitary i/o devices. Hardware innovations of the TX-2 include: addressable magnetic tape for a filing system; The Lincoln Writer, a typewriter for engineering/scientific use; and the TX-2 multiple sequence operation for rapid context switching. While the TX-2 was initially a personal computer, it operated under control of a timesharing operating system by the mid 60's. The TX computer circuitry was virtually identical to the logic and laboratory modules that DEC sold in its first four

years, prior to the introduction of the PDP-1 in 1961.

These machines, used as personal computers, pioneered numerous applications, the most famous being Sutherland's Sketchpad.

I spent a brief but wonderful period (1/59 - 6/60) using TX-0 for speech research and writing the program to do "Analysis by Synthesis", (Bell, et al 1961) which took as input speech spectra directly from a filter bank connected to TX-0. I also spent six months designing an IBM-compatible tape TX-0 and exploring hybrid computation (an analog computer used with TX-0). Peter Deutsch, then age 12, acted as a relatively patient user consultant and helped me debug my macros. Gwen Bell entered data on land use in the Boston area, and I wrote a program to display and explore this data. The program was demonstrated to the city-planning faculties at MIT and Harvard. Ten years later, Harvard was able to turn the clock back over 30 years when they established their computer graphics laboratory to do a simpler version of computer mapping, using punched cards and line printers.

DEC PDP-1 DISPLAYS: TYPE 30, 31, AND THE FIRST COLOR DISPLAY

The PDP-1 continued in the tradition of the "MIT personal computers". The PDP-1 had three displays. The Type 30 point plotting display, which took 50 microseconds to display a point on the 10" x 10" tube with 1K resolution. This was one of the first displays connected to a commercial computer, converting it into a personal computer for various scientific data analysis applications. Type 30 was extended to include a character generator ~~which was~~ controlled by loading a 36-bit, 5 x 7 bit raster specifying the character. The circuitry, designed by Ben Gurley, formed the basis of the 3XX displays. The DEC Special Systems Group designed an elaborate display, PEPR, for bubble chamber photograph analysis for MIT using the basic display circuits.

Type 31 designed by Gurley was a 5" high precision display with 4K point resolution. Only two? were built, the first being delivered to Lawrence Livermore Laboratory (LLL, now LLNL--the "N" for National) for use on their PDP-1. The Livermore PDP-1 had every option that DEC had proposed to build including Remington Rand tapes, and was used as a front end and exloratory PC for the large computers including Stretch and LARC. Ben Gurley left DEC and with Ed Fredkin established Information International Inc.(III). The first III product included a high-resolution display for various image input/ouput applications. The basic high resolution display is still in existence.

Two? PDP-1 Color Displays were built, one of which was delivered to the Air Force Cambridge Research Laboratory (the military counterpart to Lincoln Laboratory). The color display was built by modifying a standard RCA color television for point plotting.

At the Fall Joint Computer Conference in 1961, Ed Jacks, from General Motors, spent almost the entire conference watching people use the PDP-1/Type 30 draw and doodle using a program I had written. The display influenced Ed to tackle Computer Aided Design, later using the IBM 7090 with a connected display. The classic program SPACEWAR! was written on the PDP-1 in 1962, by Steve Russell. Today SPACEWAR! continues to be the center to settle litigations about the design of computer games.

LABORATORY PC'S: LINCOLN LABORATORY LINC, DEC PDP-12, AND MINC

Wes Clark and Charles Molnar continued the MIT tradition of building personal computers, and LINC was designed for wider scale use for the life sciences, by being lower cost (about $40K in 1962). The LINC was designed using mostly DEC modules. The original LINC had two 5" displays, a keyboard, 1 or 2Kw, 12-bit memory and two LINCtapes. Each unit had approximately 256 Kbytes of addressable tape memory. I believe LINC was the first production personal computer for scientific use (about 50 were produced, 21 by DEC). Its significance was its completeness at the low cost, yet being open-ended.

DEC went on to build 140 LINC-8's beginning in 1967 (designed by Wes Clark and Dick Clayton) and around 1,000 PDP-12's beginning in 1970 (Bell, Mudge, and McNamara, 1978).

DEC 338, 339, AND 340 DISPLAYS

The 338/PDP-8 display/personal computer, introduced in 1967, was used as a front-end computing terminal. The 338 and 339 (for the 18-bit computers) was, I believe, the first display processor (Bell, Mudge, and McNamara, 1978). It took a display program in the PDP-8's primary memory, which specified points, character strings, vectors, subroutines and other program-control instructions, and permitted completely autonomous operation of the display, independent of the central processor. The 339 was used with the 18-bit computers, especially PDP-9. Both systems were used for standalone data analysis and for display terminal front ends to large computers.

A special version of the 340 was designed on a Saturday morning with Ivan Sutherland for his use within the government just prior to joining ARPA. This PDP-7/340 had every conceivable keyboard and input device we could think of, including switches, trackballs, joysticks, lighted buttons, ...

DEC PDP-15/VT15, AND GT4X "WORKSTATIONS"

The PDP-15/VT15, introduced in 1970, costing about $90K was used as a standalone for one or two users (to reduce the cost to $50K per terminal using a $15K graphics terminal). About 30% of the systems had two terminals. It took about 100 microseconds to draw a vector using the DDA technique. The PDP-15 had from 8 Kw to 32 Kw of memory. Computer Aided Design system for Printed Circuit layout written and marketed by REDAC (U.K.) was a key application, although it was used extensively for scientific-research applications.

In January 1973, the GT40 was introduced at a price of $25K with 16 Kw of memory based on the lower cost PDP-11/05. The well-known program, Lunar Lander, was written by Jack Burness for the product introduction. The relatively slow speed of the display, coupled with a limited memory and configuration of the 11/05 limited the application to a pre-processor for time-shared systems doing CAD, etc.

The GT44, was introduced in June 1973 at a price of $38K, using the same display processor as the GT40, but with an 11/34 computer and RK05, 5 Megabyte disk. The GT44 was used as a complete, standalone computer.

The GT48, introduced in October 1975, selling for $55K, was a high performance personal computer, using the 11/34, with memory up to 128 Kw. Vectors were generated using analog circuitry at a rate of 100,000 vectors per second.

THE PDP-8 WORD PROCESSING PERSONAL COMPUTERS
In addition to the early use of the PDP-8, with the 338 and various storage tube displays, the PDP-8 began to be used extensively 10 years after its birth for the DECmate series of word processing terminals. The PDP-8 was based on the PDP-5 (1963) 12-bit architecture, and influenced by the CDC 160 and LINC, addressed up to 32 Kw (later expanded to 128 Kw). The following word processing personal computers used the PDP-8:

Year	Model	Price	Comments
1975	DS310	$16K	8/A, desk, 8" floppy and VT52 terminal,
1976	WS200	$40K	8/A, cabinet, 1-8 users, 5 Mbyte disks,
1977	VT78	$15K	Intersil 6100 PDP-8 chip, embedded in VT52, 8" floppies
1981	DECmate I	$12K	6120 chip, embedded in VT100, 8" floppies
1983	DECmate II	$3.5	integrated unit, 5" floppies, bit mapped,
1984	DECmate III	$2.4	Z80/8086 and 5" winchester disk options

By 1984, the 21st birthday of the basic architecture, over 100,000 PDP-8 connected terminals or personal computers were used for word processing. More PDP-8s were built in 1984 than any other year. In a very large fraction of the use, the personal computers connected with larger departmental-level VAX computers for electronic mail, filing and auxiliary word processing for a true, shared workstation environment.

LSI-11 BASED PERSONAL COMPUTERS: TERAK, PDT 11/150, AND VT103
The LSI-11 (11/03), introduced in 1976, with a smaller printed circuit board form factor (approximately 10" square) and using the Qbus proved to be an important building block for Digital's subsequent personal computers.

Terak was founded in 1975 to exploit the notion of a bit-mapped graphics computer and used the LSI-11 (11/03) and subsequently the 11/23. The 8510, first delivered in 1976 to Ken Bowles at the University of California, San Diego for the development of UCSD Pascal, was developed independent of the Xerox Alto. The system operated with up to 28 Kw of memory, used 256 Kbyte floppies, and the bit-map display's resolution was 320 x 240. About 3000 were built and sold at a price of $5000 to the education market.

DEC introduced the PDT 150 as a small computer with two 8" floppies and 32 Kw at a price of less than $10K. While all the DEC marketing groups debated the price and their role in distribution, the personal computer industry formed. DEC was able to discourage Dan Bricklin from using PDT to build Visicalc. (He went on to use the newly introduced Apple II.) The VT103, introduced in 1981, was based on the VT100, which had been introduced to accept the Qbus, LSI-11/23 modules.

DEC TYPESETTING TERMINALS: VT20 AND VT71
The PDP-11, introduced in 1974 and 1977, was used as a base for the VT20 and VT71 typesetting terminals. These had a programmable character set of 256 characters, used 15" portrait oriented tubes, and communicated with the host via 9600 baud asynchronous communication. The VT20 was based on the 11/05 and could handle two CRTs, while the VT71 used the 11/03 and sold for $8 K. These terminals connected to host PDP-11's which carried out the main typesetting editing functions.

DEC VAXSTATIONS: 100, I, II AND 500 SERIES
The introduction of high resolution, bit-mapped terminals connected to VAX occurred quite late in VAX's life. A number of attempts were made, including SUVAX (for single user VAX) providing both monochrome and color, operating in 1980.

The VAXstation 100, introduced in May 1983, was a high resolution 19" terminal with bit-mapped graphics, which connected to any of the VAX computers by a high speed fiber optic link connected to the VAX's I/O bus (the Unibus).

VAXSTATION I and II, based on the MicroVAX I and II, were introduced in October 84 and June 1985, respectively. Both units sold for approximately $20K, with 19" bit-mapped CRT, using the Qbus for interconnecting I/O equipment.

The VAXstation 500 family was introduced in October 1985 with 1280 x 1024 pixel resolution color permiting 2D and 3D wire frame and 3D shaded solid images.

CONCLUSIONS
Personal workstations have evolved from many directions and along many dimensions. This conference barely scratches the surfaces of a few dimensions. We should endeavor to be as complete and accurate as possible in what we provide and in going from these papers to a more widely available publication.

The conference has served to convince me that the critical dimensions of personal workstation evolution should be explored and documented by historians.

ACKNOWLEDGEMENTS
Dave Fanger, Encore Computer, did the figures on a Xerox Star. Designers of DEC products supplied product information: John Clarke and John Kirk (PDP-8), Herve Lavoie (VT15 and GT4x), Jim Milton (typesetting), and Stu Weckar (DECnet). Dave Nelson, Apollo, contributed a table and much to my understanding of workstations. Gwen Bell helped clarify my thoughts well enough to get a first draft, which she then edited. Mary Jane Forbes prepared the final manuscript.

APPENDIX 1. THE DIGITAL VAX, HOMOGENEOUS COMPUTING ENVIRONMENT

TECHNOLOGICAL AND MARKET BACKGROUND

BATCH MAINFRAMES FOR CENTRAL SERVICES

In the first two computer generations, 1950-1970, computers were used in batch processing under the name of mainframe computing. During the 70's the mainframe began to be used almost interactively from remote job entry terminals as "glass key punches". The general direction is to have larger mainframes and larger terminal networks that interconnect to a single computer by an array of front-end computers. When more power is required, more switching computers are connected to several mainframes each of which perform a particular function. Attached dual processors are used to provide increased power for what is fundamentally a single system. Over time the evolution will be to small-scale multiprocessors for incremental performance and higher availability.

MINICOMPUTERS AND TIMESHARING FOR A GROUP

In the mid-60's both minicomputers and timesharing were developed at Digital around the PDP-8 and PDP-10 computers respectively to lower the cost of computing. Minicomputers were initially used as components of real-time systems and for personal computing. The LINC minicomputer, developed at M.I.T.'s Lincoln Laboratory, was the first personal computer, providing a personal filing system and the ability to write and run programs completely on line.

Timesharing started out as a centralized mainframe facility for a large group with the early demonstration at MIT with CTSS, followed by the introduction of products by DEC (PDP-6), GE (using Dartmouth's BASIC), and Scientific Data Systems 940 (based on the Berkeley Timesharing System). Access was via individual teletypes which were eventually replaced by cathode ray tube terminals, or "glass teletypes". By the mid-70's low cost PDP-11 timeshared computers began to be used by separate groups and departments to provide "personal computing". By the mid 70's a number of minis used high performance graphics terminals for CAD/CAM, typesetting and other applications requiring graphical output or fast response. These minis were the forerunner of today's distributed, personal workstation environment.

In the early 80's, low cost disks and large memories permitted two evolved computer structures: the 32-bit supermini, and the microprocessor based "team computer". The supermini had all the power of its mainframe ancestors, especially the critical 32 bits to access memory. The "team computer" based on modern, powerful microprocessors, is simply much lower priced (eg., $15,000) providing "personal computing" at a price below personal computers.

WORKSTATIONS

The fourth generation appeared in 1972 with the microprocessor. With the second 8-bit generation microprocessor, floppy disks and 16 Kilobit semiconductor memories (circa 1976), personal computers were practical and began to be manufactured by Apple, Commodore, Radio Shack, etc. With 16-bit microprocessors (measured by data-path) and 64 Kbit rams, the second generation of PCs appeared in the early 80's which were suitable for building personal workstation environments.

Table 4 summarizes the characteristics of these styles of computing.

In 1979, Carnegie-Mellon University wrote a proposal for personal computer research to implement a workstation environment, and declared: "The era of timesharing was ending." Today's powerful personal workstations, such as the Apollo or SUN workstation, providing a large virtual address, connected with shared facilities on a local area network, characterize this type of machine and computing environment.

PERSONAL COMPUTERS CLUSTERS (I.E., THE WORKSTATION ENVIRONMENT) AS AN ALTERNATIVE TO SHARED COMPUTERS

In the mid 70's, Xerox Research Parc researchers developed and provided themselves with a personal computing environment consisting of powerful personal computers all linked together via the first Ethernet cable (3 Mbits), and created the notion of the Local Area Network. Their network had various specialized function servers, including a shared central computer that was compatible with the DECsystem 10, for archival memory and large scale computation.

Figure 13a shows the hardware and software of a multiprogrammed computer used for timesharing, and the corresponding structure of a personal computer cluster consisting of functional services and interconnected by a common interconnect that provides basically the same capability. The timeshared system has a central memory containing various jobs connected to terminals and operating system which attends to the users and handles the particular functions (eg., real time, files, printing, communication). Personal computers are connected to timesharing systems as terminals. By comparing the shared system with the systems formed

Table 4 - Computing Style Characteristics (Circa 1978, G.Bell)

STYLE*	CENTRAL	DEPARTMENTAL GROUP/TEAM	PERSONAL
machine	mainframe	mini	micro
price range	500K-5M	10K-500K	1K-10K 10K-50K**
communications	coupled to terminal network	to mainframes, peers and PC's	terminal emulation
database	organization's archives as service	organization, project function	personal files
terminal access	"glass keypunch"	"glass Teletype"	direct CRT
typical uses	corporate accounting, electronic mail	project CAD, order entry, small business "mainframe"	word processing, financial analysis, CAD**
caretaker	a staff providing service	distributed with user group	the user

Comments (1985):
* Not including regional-styles supercomputers
** Workstations used on a one at a time basis.

from functionally independent modules, one would expect two design approaches:

1. decomposing systems to provide shared LAN services, and
2. aggregating Personal Computer to Form PC Networks and Clusters.

Decomposing Systems to Provide Shared LAN Services

As shared computers become more complex and more centralized, it's desirable to decompose the functions for execution on smaller computers that can be distributed to be nearer the use. Thus, the decomposition of a shared system into various boxes, each of which perform a unique function permits the evolution of the parts independent of the whole, the physical distribution of a function and the ability of several computers to share a function (see Figure 13b). While we have described the evolution of Local Area Networks (LANs) as a decomposition of a single system, LANs are generally an aggregate of heterogeneous systems which access a shared service of some kind as described below.

LANs differ from Wide Area Networks (WANs) in that they assume a low latency, high bandwidth interconnect. This permits file access as well as file transfer applications. With file access, it is possible to remotely locate part or all of a system's mass storage to a file-serving computer. File access requires bandwidth and latency which are roughly equal to that of a disk (i.e., 10 Mhz rates); file transfer can be done at substantially slower rates (56Khz to 1 Mhz).

Using the reasoning which allowed the formation of the file server, we continue the decomposition of a large central system into servers or stations and then combine these servers into a LAN. The major servers include:

1. Person Server (the personal computer used as a workstation) - local computation and human interface, possibly private storage of files
2. File Server - mass storage
3. Compute Server - batch computation or existance of particular programs
4. Print Server - printing and plotting of graphics images
5. Communication Server - terminal, telephone and PABX, Wide Area Network access including interface to international standards, other companies (eg. SNA)
6. Name/Authentication/Directory Server - naming the network's resources and controlling access to them.

A LAN formed as a complete decomposition of a single system and containing no other incompatible servers would be defined as a homogeneous cluster of Personal Computers or Workstations.

Aggregating Personal Computers to Form PC Networks and Clusters

As personal computers require more facilities (e.g. printing, communication and files), and the number and type of PCs grow, the need to directly communicate for sending messages and sharing files. Furthermore, as a collection of computers in one place forms, economy is gained by sharing common facilities such as printers, phone lines, and disks. Applenet and Corvus Omninet are relatively short and low data-rate Local Area Networks used to permit the construction of what might best be called a network of personal computers because of the heterogenity of type. The 3Com system for interconnecting IBM PCs is more characteristic of the homogeneous network, or cluster.

For a PC cluster, one would expect to have a single

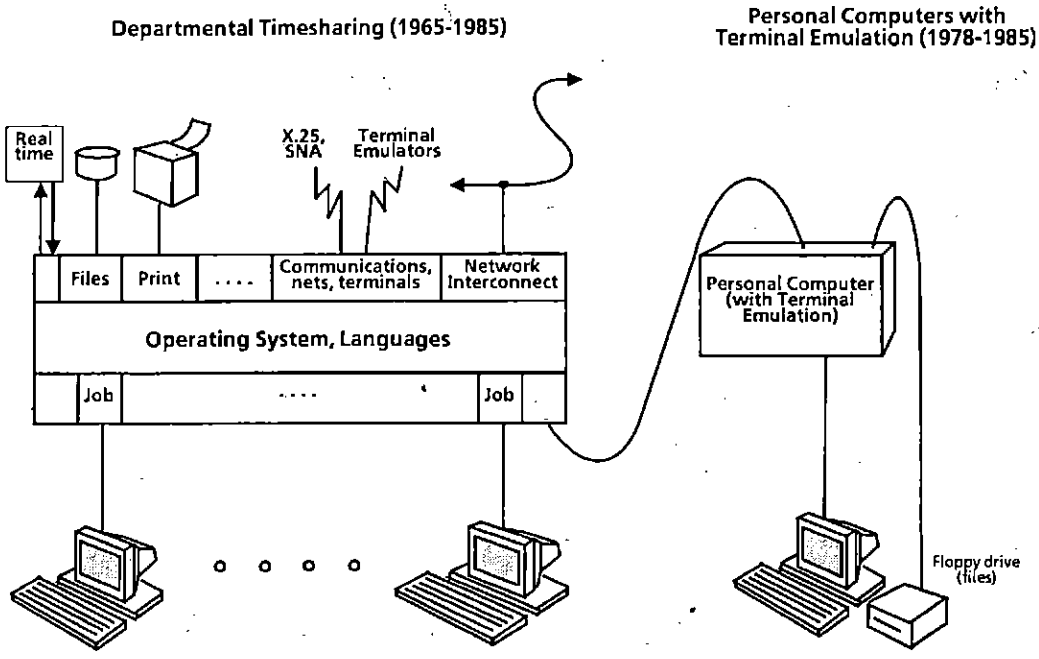


Figure 13a. Departmental timesharing (1965-1985) and personal computers with terminal emulation (1978-1985).

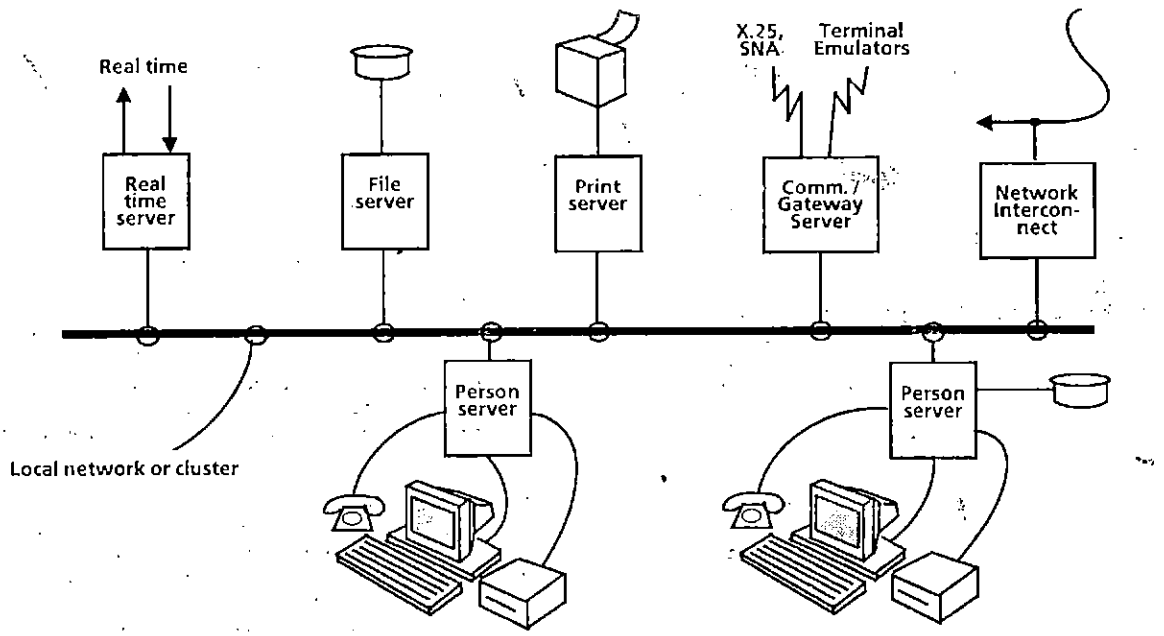


Figure 13b. Personal workstation computing environment using LAN clusters (1980's).

File Server which can supply records at random to any of its constituency. Table 5 summarizes what timesharing, PC's and PC Clusters provide.

DISTRIBUTED PROCESSING USING CAMPUS AND WIDE AREA NETWORKS

The proliferation of timeshared computers required the development of networking in order for various systems to communicate with one another and to mainframes. Thus, dispersed computing became distributed computing. Store and forward wide area networks evolved from the ARPAnet, which was used to interconnect timeshared mainframe computers (mostly PDP-10's).

Table 5 - Timeshared, PC, PC Cluster and PC Network Characteristics (Circa 1981, G.Bell)

WHAT	TIMESHARED SYSTEM	PERSONAL COMPUTER	PC CLUSTER*/ NETWORKS
processing	highest peak	lo-med, guaranteed	= PC
programs size	very high peak	small to medium	= PC
filing	large	small, guaranteed (+ off line)	= PC and TSS
communication	network	term. emulation	= PC and TSS
CRT	slow response "glass Teletype"	fast response, screen oriented	= PC = PC
cost	fixed, can go to lowest$/terminal	lowest entry	f(no. of PCs)
secure	shared, public access	totally private	contained/TSS
pros	explicit costs shared programs, large jobs	low entry cost, "owned" by indiv., security, SW publishing = low cost	ability to expand, shared facilities, better match to org. structure
cons	shared, poor response for terminals, higher entry cost, security	limited capability, but increasing most rapidly	limited proc/prog., shared facilities
heterogenity	one homogeneous system	micro computers	one system/ can be heterogenous

*Personal workstation environment

Campus Area Networks

When a collection of Local Area Networks are connected together in a single area which extends beyond a typical LAN, we call this a Campus. Universities clearly typify the campus as does a collection of buildings. Gateways are used to interconnect LANs of different types (eg, Omninet, Ethernet (802.3), IBM Rings (802.4), Applenet, Arcnet, Broadband token bus (802.2) PCnet), whereas bridges or repeaters are used to interconnect networks of the same type to form one larger network. Bridge technology is necessary in connecting multiple LANs together on a wide-area basis using high-speed links (e.g., satellites).

Wide Area Networks

WANs are characterized by low bandwidth (up to 56 Kbits), high latency, errorful transmission, and autonomous operation of the nodes. The applications typically include: mail, file transfer, database query, and low interaction remote terminal accesa. Wide Area Networks can be constructed in several ways: direct dial up using conventional circuit switching with voice-grade circuits, an intermediate store and forward network such as Telenet, or a hybrid approach where various worker computers do store and forward switching.

THE VAX, HOMOGENEOUS ENVIRONMENT OR "E"

Although the specific design of the VAX Environment began in December 1978 with the approval of the Corporation, its origins include:

- the original VAX-11 goals for a 1000:1 range of computers;
- evolution of distributed processing minicomputer networks, in Wide Areas, "Campuses", and Local Areas;
- the appearance of powerful Personal Computers and Local Area Networks, permitting the aggregation of tightly coupled "PC networks and clusters" that provide some of the benefits of timeshared minicomputers and mainframes;

- the ability to aggregate minicomputers and mainframes into multiprocessors (the 78.2) and multicomputer clusters (VAX clusters) that appear to be a "single" system in order to provide higher reliability, higher performance, and incremental performance.

The December '78 statement of the Distributed Computing Environment (Figure 6) and subsequent evolution [shown in brackets] was:

> "Provide a set of homogeneous, distributed-computing-system products based on VAX-11 so a user can interface, store information and compute without re-programming or extra work from the following computer's system sizes and styles:
> - via [a cluster of] large, central (mainframe) computers or network;
> - at a local, shared departmental/group/team (mini) computer [and evolving to a minicomputer with shared network servers];
> - as a single user personal (micro) computer within a terminal [and evolving to PC Clusters];
> - with interfacing to other manufacturer and industry standard information processing systems; and
> - all interconnected via the local area Network Interconnect, NI [i.e., Ethernet] in a single area, and the ability of interconnecting the Local Area Networks (LANs) to form Campus Area and Wide Area Networks."

Figure 7 shows the origin of the "E" shape that characterizes the VAX Homogeneous Computing Environment. The three horizontal segments of the E provide the different computing classes which roughly correspond to different-priced computers; the functions are described in Table 4. In order to implement the environment, many requirements were initially posited, and several developments evolved from necessity:

- a range of VAX-11 and 11-compatible computers to meet the requirements of the various computing styles based on different classes of computers;
- interconnection schemes and the corresponding protocols for building multiprocessors, tightly coupled centralized VAX Clusters, LAN-based PC Clusters, LANs, Campus Area Networks and Wide Area Networks;

GOALS OF THE ENVIRONMENT

PRODUCT RANGE

The important goals and constraints of the Environment are contained in the original statement about what the Environment, which is simply "to provide a very wide range of interconnectable VAX-11 computers." The original goal of VAX was to be able to implement the range [for what appears to be a single system] of a factor of 1000 price range... with no time limit given. Since a given implementation tends to provide, at a maximum, a range of 2-4 in price and 10 in performance if performance is measured as the product of processor speed times memory size, then many models and ways of interconnection were required.

At the time the 780 was introduced, the total range of products for both the VAX-11 and 11 family was almost 500, beginning with $1,000, LSI-11 boards and going to a $500,000 VAX-11/780. The VAX 8600 increased the performance range by another factor of four. If the LSI-11 is included as a personal computer, the price range is reduced to only a factor of 50. While the two ends of the system were "compatible" and could be interconnected via DECnet, they lacked the coherency necessary for a fully homogenous computing environment.

By introducing "VAX Clusters" (Figure 7), the range can be extended by a factor of up to the number in the cluster, or about a factor of 10 more. For VAX, Digital now provides a price range of from about $20,000 for a MicroVAX-II to about $7.5 million for a cluster of twelve 8600's and a corresponding performance range of several hundred. The VAX Cluster, shown at the highest level of the hierarchy appears to the user as a single, high availability system.

The VAX Cluster work was initiated concurrent with the VAX Strategy to address the fault-tolerant, and incremental expansion needs for shared computing including a common disk and file system for the cluster. The interconnection of the clusters, using the Computer Interconnect, CI, was the beginning of a standard's activity for interconnecting computers on a standard basis.

STATIC AND DYNAMIC ASSIGNMENT OF PROGRAMS TO NODES

Ideally, a user can decide on how to compute on a completely variable basis at the following times:

- at system purchase or rent-time ranging from outside facilities reached via gateways, to a central facility, to a shared department or team computer, to a user's own personal computer.
- at system-use time, ranging from access via a terminal, or personal computer interconnected to the system LAN, or a particular shared computer. Here, work is bound statically to a particular set of system resources. Most likely, particular nodes would execute special programs on data located at the node.
- at task time on the basis of reliability. VAX clusters provide for dynamic allocation of work among the computers in the cluster to effect load balancing since files are centralized.
- at task-use time on a completely dynamic basis, ranging from computing on his own local system to being able to collect any resources and move work dynamically while programs are in execution. With this ability, as a program goes though its various stages of development, it might be moved from small system to large system to take advantage of increased computational power at higher level nodes.
- at task time on a dynmic basis with the ability to acquire arbitrary resources to engage in parallel computation.

CONCLUSIONS

The VAX Homogeneous Computing Environment is an important computer structure for providing a computing environment. This environment encompasses all styles, sizes, and computing classes, from the traditional mainframe and the high availability cluster (utilizing a fault tolerant architecture) to distributed minicomputers, and now the distributed personal workstations. It was stimulated by a number of factors including the distributed-workstation-environment idea. In terms

of architectural scope, it is much broader than simply a range of computers as typified by the evolving IBM 360 -> 370 -> 43xx -> 30xx series because it provides full compatibility through homogeniety for a complete range of computing styles and not requiring separate architectures for shared-minicomputer systems (System 36, System 38, and Series 1), personal computing (IBM PC, word processor) or terminal switching and gateways (e.g. IBM 8100).

Note: this Appendix is solely the view of the author and is not necessarily the view of Digital Equipment Corporation or its engineers.

REFERENCES

Bell, C.G., "The Multi: A New Class of Multiprocessor Computers", Science Vol 228, No. 4698, 26 April 1985, 462-467.

Bell, C.G., H.Burkhardt, S.Emmerich, A.Anzelmo, R.Moore, D.Schanin, I.Nassi, and C.Rupp, "The Encore Continuum: A Complete Distributed workstation-Multprocessor Computing Enviroment", AFIPS Conference Proceeding, 1985, Vol 54, 147-155.

Bell, C.G., G.Fujisaki, J.M.Heinz, A.S.Stevens, and A.S.House, "Reduction of Speech Spectra by Analysis-by-Synthesis Techniques", The Journal of the Acoustical Society of America, Vol 33, No. 12, 1961, pp1725-1736.

Bell, C.G., J.C.Mudge, and J.E.McNamara, Computer Engineering, Digital Press, Bedford, Massachusetts, 1978.

Bell, C.G. and A.Newell, Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971

Burger, R.M., R.K.Caving, W.C.Holton, and L.W.Sumney, "The Impact of ICs on Computer Technology", Computer, Vol. 10, No. 10. October 1984, 88-96.

Clark, W.A. and C.E.Molnar, "The LINC: A Description of the Laboratory Instrument Computer", Annals of the New York Academy of Sciences, Vol 115, July 1964, 653-668.

Metropolis, N., J.Howlett, and R.Gian-Carlo, editors, A History of Computing in the Twentieth Century, Academic Press, New York, 1980.

Pake, G.E., "Research at Xerox PARC: A Founder's Assessment", IEEE Spectrum, Vol. 22, No. 10, October 1985, 54-61.

Perry, T.S., and P.Wallich, "Inside the PARC: The Information Architects", IEEE Spectrum, Vol. 22, No. 10, October 1985, 62-76.

Ralston, A. (Editor) and E.D.Reilly (Assoc. Editor), Encyclopedia of Computer Science and Engineering, Second Edition, Van Nostrand Reinhold Company, 1983, New York.

Redmond, K.C. and T.M.Smith, Project Whirlwind: A History of a Pioneer Computer, Digital Press, Bedford, Mass., 1980.

Roberts, L.G., "Computer Network Development to Achieve Resource Sharing", AFIPS Conference Proceeedings, SJCC, Vol 36, 1970, 543-549.

Siewiorek, D.S., C.G.Bell, and A.Newell, Computer Structures, McGraw-Hill, New York, 1982.

Wexelblat, R.L., editor, History of Programming Languages, Academic Press, New York, 1981.

A PERSONAL VIEW OF THE PERSONAL WORK STATION
SOME FIRSTS IN THE FIFTIES

By

Douglas T. Ross
SofTech/MIT

Section 1

INTRODUCTION

"What the h--- kind of program do you have there?" -- the intercom blasted forth, the sound distorted by the intensity of the shout. It was the chief computer operator in Test Control (as the standard control area for the Whirlwind I Computer was called) calling me in great agitation. I couldn't reply at once, for I was some twelve feet away from the loudspeaker of the E31 console -- in the far back corner of the secret and darkened Room 222 of the Barta Building at MIT. I was hunched over a 16 inch oscilloscope called the Area Discriminator. It was mounted vertically in a box on the floor, so its face made a horizontal flat surface. Adhered to the tip of my moving finger was a bright, glowing displayed spot of blue-white light, about 1/4 inch square. I was in the process of writing my name into the computer with that spot -- freehand. It was sometime in the Fall of 1954.

It was one of the few programs I ever wrote that worked the first time. I remember that I dreamed it up on a long flight back from Texas, which must have been at most a few days earlier. Fewer than 200 instructions were required, and the hardest part was understanding the octal constants that calibrated the scope display coordinates.

1.1 My Sources

This is my second foray into the tribulations of writing (rather than making) history. In 1977 I worked for months on my paper Origins of the APT Language for Automatically Programmed Tools for presentation at the History of Programming Languages Conference June 1-3, 1978 in Los Angeles. (Ross 1977) As was the case for that paper, I have taken as my primary source my extensive personal collection of archival working papers, reports, and records which I have retained over the years, and I have attempted to include only observations I can directly support from these records. Unfortunately I am less sure than was the case for APT, whether or not my current story is complete, however, for the early portions hinge on older material that is less organized and perhaps less complete. I say "perhaps", because I cannot determine whether certain materials I remember seeing are missing, or merely are misplaced. At one point in SofTech's impecunious past, rather than buying needed file cabinets, all of my historic files were without my knowledge summarily crammed into a large number of storage boxes whose length dimension was more than one file drawer. I still have not been able to retrace what was where, as useful chinks in the boxes were filled here and there by segments of unrelated files! In later years I may be able to fill in more details, as relevant portions come to light, but even so, what I have found already yields an engrossing trail.

As in the APT history paper, to provide concise reference to the materials in my files (which ultimately will reside in the MIT Archives) without spelling out each item I have used the following condensed notation: [C540123] means "correspondence 1954 January 23"; [R56123] means "daily resume (a form of personal professional diary) 1956 December 3"; and [WW2Q55p10] means "Whirlwind I Quarterly Report Second Quarter, 1955 -- page 10", see (Project Whirlwind 1952-1957) in the list of of References.

1.2 The Theme

I am not a historian at all, and can't bring myself to write a straightforward (dull) recounting of events. For these stories from the past I feel I must have a theme to provide some perspective. Also, by writing in the first person, I can try to give some of the feel of the times in ways that I hope add interest. But also I think such a style allows a more accurate interpretation of what actually was in the air as the documented events took place, even though

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0019 $00.75

surely all that is written is biased by recollections, as well.

I take the unique event of the first hand-drawn input to a computer as the starting point of my theme for this paper -- that The Personal Work Station as a working reality (rather than a speculative idea) did have a recognizable era of beginning more than thirty years ago. Like the present era, where the physical and performance characteristics of the work station aspect of the PWS are so essential, the fact that Whirlwind had the Area Discriminator scope was an essential ingredient of my successful program. But my theme is much sharper than that, and in a sense even contradicts the natural emphasis on the work station characteristics which dominates most writing on the topic. My real theme is that it is the revolution in personal work (i.e. explicitly the PW, rather than the WS, of the PWS) that was, and is even more so today, what is important. (This also is why, throughout this paper I purposely spell PWS with three words rather than the accepted (correct) two -- to stress my view.)

I hope to show that from the very beginning, the changes in personal working style, made possible by the idea that man and machine can share in the problem-solving process, is what makes the Personal Work Station idea a unique and valuable departure from the customary view of using computers. With the PWS idea, the computer becomes a partner not a mere tool. And to integrate and actually realize this PW and WS idea, requires a systematized treatment of the software that links man to machine. That is the third component of my theme.

1.3 Overview

In the time available to me, I can trace only the beginnings of this theme. As was the case with the APT story, I find that my records show that this PWS story breaks "naturally into a number of overlapping but nonetheless distinct periods during which the focus of attention highlighted first one and then another and then another aspect of the overall subject." (Ross 1977, p. 282.)

The first period covers my introduction to Whirlwind and programming, in the Summer of 1952, and brings out that even with all its rooms full of equipment in its own building at MIT, Whirlwind was a miniscule microcomputer by today's standards for personal work station capacity. And to start with, Whirlwind lacked some needed features, as well.

The second period, from August 1952 through August 1953, saw the flexibility and capacity of Whirlwind expanded, as facilities geared up for the focus on the development of the Cape Cod System -- the R&D base that led to the full-scale development of the SAGE Air Defense System by Lincoln Laboratory. (Redmond and Smith, pp. 201-213.) Most of the environmental equipment for PWS evolution now was present, but my available records don't indicate whether or not I yet had discovered the wonders of the mysterious Room 222.

In any case, well before the Summer of 1954, during the third period when I completed my Masters thesis (Ross 1954b) and our work on Large Scale Data Reduction programs (Ross 1953b) actually got under way (which was the driving force for my own insights and endeavors), I did gain access to the classified areas of the Barta Building and knew that we simply had to incorporate the marvels of manual intervention (or MIV as it was then called) into our system capability. The problem was that I didn't know how to program for all that equipment.

The problem was solved in the fourth period (June 1954 through early 1955) by hiring Bill Wolf from Lincoln Lab. Bill had done some of the early MIV programming for Lincoln. His early departure forced me to learn what I needed to know as I completely rewrote his code to conform to my plans, which were more general than Lincoln's use of the same WS equipment. This also, of course, was the period when I wrote the Scope Input program. (Ross 1954d) Another innovation triggered by my needs was the creation of the Director Tape utility program [WW3Q54p7] -- the first real Operating System command language system (to use present-day terms) to eliminate the computer operator function for my elaborate, multi-tape runs. By the third quarter of 1954 my plans for MIV use were quite general and well developed.

The fifth period focuses on the preparations for and delivery of two Servo Lab symposia at MIT. The first, held on March 8 and 9, 1955 on "Design and Evaluation of Bomber Fire-Control Systems" (MIT Servo Lab 1955) covered our project's entire hardware and software system for complex system testing. The second, on June 1 and 2, 1955, covered just the MIV-Controlled Data Reduction Methods. (Ward and Ross 1955) Both symposia demonstrated the new techniques to large groups from outside MIT. My PWS ideas were now a working reality, in this particular setting.

The sixth period concerns my first professional paper, Gestalt Programming: A New Concept in Automatic Programming (Ross 1956b) which I presented as the opening paper at the Western Joint Computer Conference in Los Angeles, February 7, 1956. (WJCC 1956) I actually had proposed the paper in November 1955, [C551117, to B.J. Bennett, Prog. Chrm.] and had presented a preliminary version at MIT on January 11, 1956, [C56015] but as I recount here, it was a long process to arrive at a reasonable formulation of the ideas in acceptable form, at the beginning. I consider that paper to be my own definitive statement of the PW theme of this current PWS paper, but written at that earlier time.

The seventh period actually overlaps periods two through six, in time, for it

concerns the long period when, with John Ward and others on the Fire-Control System Evaluation Project, we formulated, proposed, designed, and then assisted in the installation of a full-scale Charactron-based MIV Console for the Univac 1103 computer at Eglin Field Air Force Base in Florida for the evaluation of the B-58 "Hustler" supersonic bomber tail turret. (Ward and Ross 1956) That the actual testing would be done at Eglin had been known since at least early 1954. I had done a special Charactron demo for the June 1955 symposium, and the MIV console itself finally was installed in 1957. [C571024, J.E. Ward to J.L. Moser, Stromberg Carlson] The Whirlwind facilities were duplicated, and their actual circuit drawings were used in the design, thanks to Lincoln Lab cooperation.

Even the Whirlwind setup was not complete as yet, however, and the eighth period mentions briefly my proposal (March 1956) (Ross 1956d) for direct Flexowriter keyboard input to Whirlwind to complete our MIV facilities. Until that time, the Whirlwind Flexowriters had been hooked up and used only for printed (or punched) output and only punched tape input. Barriers to the Gestalt Programming concept, both conceptual and physical, were very high in those days, and even when the Flexo Input was available, its use was a far cry from the central role that keyboards play in today's PWS and word processing practice. We lacked the computing capacity for full on-line keyboard use.

The Flexo Input was just one part of the facilities incorporated into the SLURP System (Servo Lab Utility Routine Programs) for large-scale, experimental program development (Ross 1958) -- which is what the Gestalt Programming concept then had evolved into. It was made possible by the luxury of the expansion of Whirlwind's magnetic core memory to 6K 16-bit words (yes, that's 12K bytes, in today's terminology!) with the cf, change fields, instruction to select any two 1K banks of words at one time. [WW3Q56p44] SLURP was the first complete interactive software development environment, with many programming, language development, debugging, and display tools all integrated into an interpretive environment with Group Control for core vs. drum storage management, and automatic logging and log playback of all MIV actions. Reminiscent of today's debate regarding menu control and how many buttons to put on a mouse, a universal "MIV Box" subroutine gave complete control over these hierarchic facilities with only Major and Minor Exit Buttons and one Wait switch for stepping and selecting from one control level to another. From today's vantage point, I can't imagine how we did all that in just 12K bytes with only a few (2 or 3?) times that much magnetic drum backup. But we did. I take SLURP to be the systematized solution, third component of my PWS theme in this paper. Development continued until the demise of Whirlwind in 1958.

The ninth period artificially cuts off abruptly at the start of 1960, to match my title and put a stop to burgeoning growth of what might be covered. Its beginning overlaps the other periods of focus, with the plans and actions to carry on MIV work on the IBM 704, 709, 7090, and 7094 computers at MIT, after Whirlwind's demise. Although artificial, this actually is a good stopping place, because other, broader themes spring from this period. Timesharing, computer graphics (both hardware and software), computer-aided design, software engineering (both language and practice), and software technology tools and methods all blossomed and flourished at MIT in the 1960s, and have direct ties to the story I present here. But those are tales for another time.

Section 2

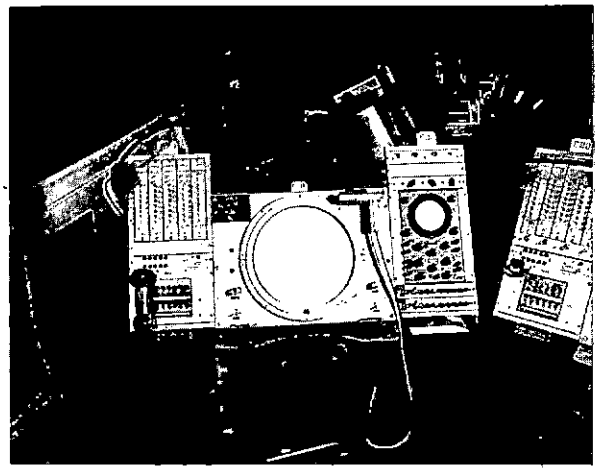
PREPARATION

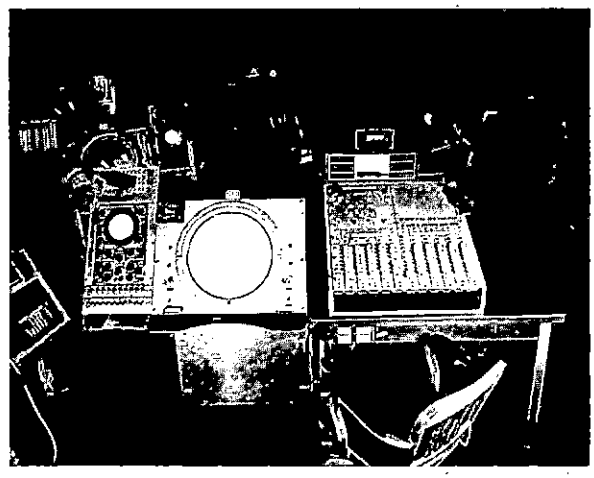
Significant events always are the rearrangement into new forms of otherwise ordinary and insignificant, routine happenings. The ordinariness allows the significance to be recognized. What was the environment, and what was going on at Whirlwind that triggered a new and more intimate kind of computer use? What were the pieces from which the PWS idea began in this story? The preparation for my theme also is the preparation stage of my story.

2.1 Programming Background

I came to MIT in the Fall of 1951 as a Teaching Assistant in the Mathematics Department, having received an AB Cum Laude in Math from Oberlin College in June. Pat and I had been married the previous January, and she was the first of a group of "computers" hired by Lincoln Laboratory (Badge No. 161) which was just getting under way -- performing calculation and graphing assignments for the engineers, using Marchant desk calculators. One of her assignments involved using a mechanical correlation computer, designed by Norbert Weiner, which was in use in the Servomechanisms Laboratory at MIT. (Ward 1954) When I needed a summer job for 1952, I applied to the Servo Lab, and was hired also as a "computer". One of my first jobs involved checking some anomalous points that had been obtained on that mechanical correlator, and someone suggested that I see if Whirlwind could be used, for they knew that Jack Arnow had written such a program. Jack later founded Interactive Data Corporation (now part of Chase Manhattan Bank) as one of the early timesharing service companies.

I, of course, had never heard of Whirlwind or of real computers, so I was fascinated. I still have Jack's little program, plus the Digital Computer Lab application form for Problem #87 (July 2, 1952) to get computer time, some scribbled yellow papers where John Frankovich (still working at Lincoln Lab) gave me my first programming pointers, and the "Tape 1414 Mod 0" July 11 printout of my Auto-correlation program. My earliest Run Request (July 18) shows that I was already up to Mod 3 one week later. (Ross 1952)





(Left) Cape Cod Console with light gun and Manual Intervention (MIV) equipment. (Right) E31 Console in Room 222. Note intervention register octal keyboard banks and intervention switches, activate pushbuttons and lights, functionally arranged. Intercom and computer restart button on right.

Before I could complete the debugging of my rather elaborate program (which later was used in many projects and theses by others) Whirlwind was completely shutdown from August 11 through August 30, 1952! [WW3Q52p6] I used the time to write a matching Fourier Transform program so that power density spectra could be computed (which also was widely used, and formed the basis for an improved program for my own Master's thesis in 1954). The reason for the shutdown is the real starting point for my PWS theme -- for in that period the entire input/output system for Whirlwind was drastically modified to provide the elaborate hardware facilities needed for Whirlwind's new and primary Lincoln Lab mission -- high priority development of an air defense system for the United States and North America.

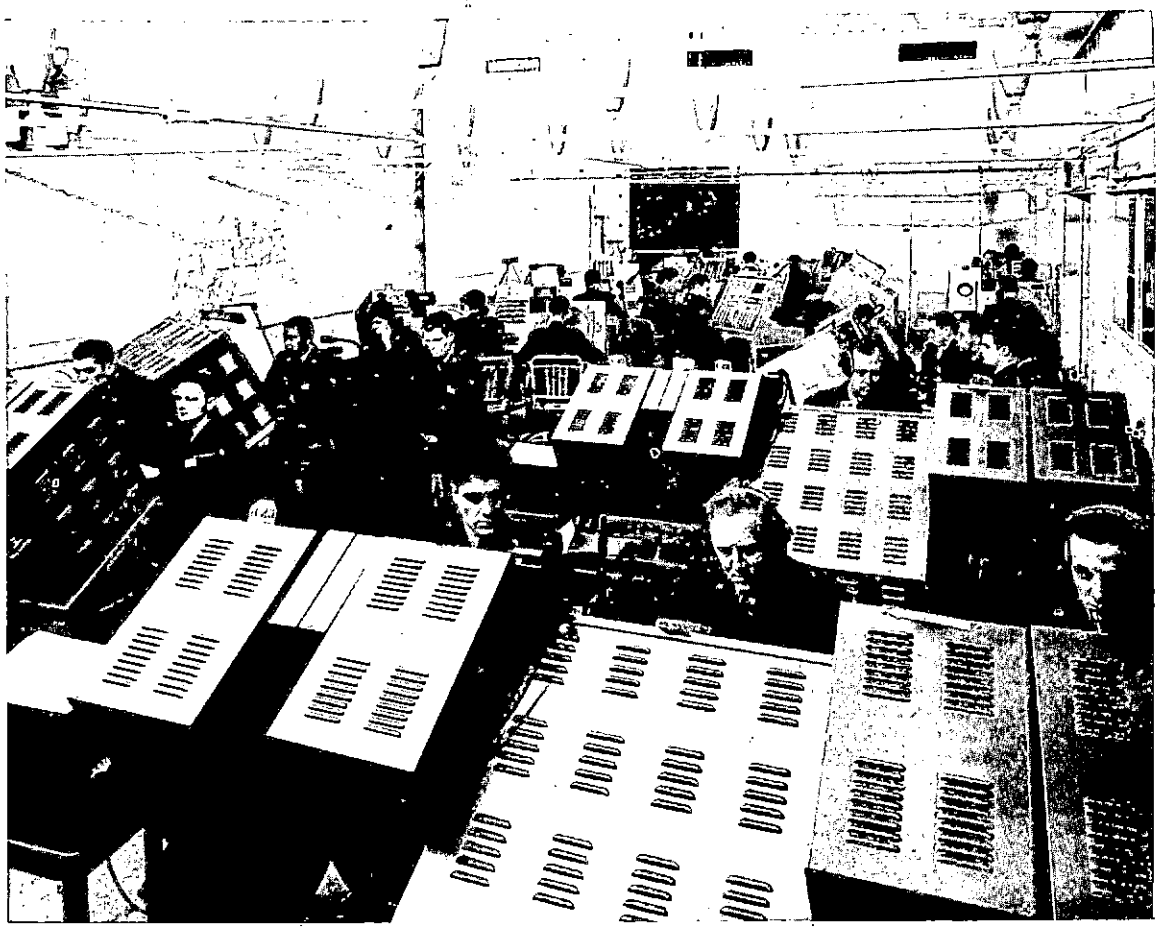
The Whirlwind order code (of 32 instruction types, maximum) was revised so that si, so, rd, rc, bi, and bo for "select input", "select output", "read", "record", "block in", and "block out" could be used for any one of a large number of input/output lines to which analog scopes, mechanical switches, "activate" (one-shot) pushbuttons, indicator lights, and magnetic drums could be attached. The mechanisms for general manual intervention in the running of programs were at hand.

Only the barest essentials trickled out to the programmers of the user problems. Most of the subsequent construction took place behind the locked, green, double doors of Room 222, which was strictly off limits. With the end of the summer at hand, and with my important programs in limbo and my enthusiasm for programming running at white heat, my supervisor, John Ward (still at MIT) asked me whether I really wanted to go back to teaching Freshman Calculus, or would I rather make a career for myself in Servo Lab, full time, taking courses as a Special Student. I felt obligated to fulfill my 1952 Fall teaching assignment, so we arranged for me to do that in addition to working full time programming and taking graduate math courses for that semester. The Math Department wasn't happy to loose me. I had been so serious and successful about by Teaching Assistant role that I had been appointed as the only graduate student member of the Dean's group of Freshman Advisors. But I remained with the Lab until August 1969, when my colleagues and I left to found SofTech.

2.2 Project Background

The Air Force-sponsored project headed by John Ward was concerned with the test and evaluation of the accuracy and performance of airborne fire-control systems -- specifically servo-controlled tail turrets in bombers. The autocorrelation and Fourier transform programs were part of the study of radar noise analysis. I was the only mathematician in the Laboratory, and one reason for hiring me was that a new form of Air-Mass Ballistic Tables had just been developed, (Nielsen and Heyda 1951, p. 9) more accurate than the old tables which had been developed for ground artillery and required extensive compensation to account for rapid movement through rarified air. My first major assignment (November 1952) was to see how these new tables could be used for the evaluation problem.

By January 12, 1953 I had a complete data flow diagram (Ross 1953a) for hit probability density calculation, based on fourteen measured inputs, and using vector calculations well suited to the three-dimensional problem and



Room 222 of Cape Cod System. Closest two airmen with backs to camera are at E31 Console (right) and adjacent console (left) used for Gestalt Programming. Directly beyond them is tripod and box of the Area Discriminator light cannon. Originally walls were black. Indirect ceiling lamps had red bulbs for system operation. (2/19/54)

digital computing. With further elaboration and refinement over the next five years, the complexity of actually programming and operating this solution for production data reduction (Ross 1953b) was the stimulus for all my MIV developments until the advent of the Computer-Aided Design Project in 1959.

The main part of the project was directed toward the development of airborne flight test instrumentation that could measure the many variables during mock attacks by a fighter aircraft pursuing a bomber. With the feasibility of the digital computer data reduction established, attention focused on digital instrumentation (primarily shaft encoders to convert selsyn-repeated analog quantities into Gray-coded binary) for recording on a flyable magnetic tape that then could automatically be read into the data reduction computer on the ground. (Telemetry was too unreliable in those days to risk on such an expensive program.) By 1954 our efforts were targeted toward meeting the needs for evaluation of the XMD-7 Fire Control System being built by Emerson Electric for the secret supersonic B-58 "Hustler" Bomber. The B-58 was not yet ready, so the program was called "Pre-B-58", and tests were run with the B-47 in the beginning. (MIT Servo Lab 1958) It was the most sophisticated test program of its time, and cost over $10,000 per flight hour, just for the aircraft, I believe.

By the Spring of 1953, programming was complete for the autocorrelation functions and work had begun on a Polynomial Fit program (Ross 1954c) to place the ballistics tables in computable form. By the summer, an elaborate Mistake Diagnosis Routine (MDR), (Ross 1953c) which allowed breakpoints to be set in an arbitrary program, with intermediate results saved for printout or displayed on the camera scope, was written and disclosed that the interpretive floating point arithmetic routines of the Whirlwind Comprehensive System (CS) were inaccurate, so they were repaired, to everybody's benefit. [WW4Q53p25andp11] The MDR played a continuing role in our experimental programming environment, especially when it later was placed under MIV control. In the Summer of 1953 the first programming for the

evaluation equations also got underway, [WW3Q53p25] but most of my time in early 1954 was focussed on my Master's thesis on minimizing Gibbs' phenomenon oscillations in the computation of Fourier transforms. But by the Summer of 1954 the programming stage was set for our first serious use of manual intervention techniques in our work.

2.3 Hardware Background

To complete the stage-setting for the Scope Input coup, I will give the following synposis of Whirlwind hardware developments during the corresponding period. In the Summer of 1952, when I started to program on Whirlwind, it had only 1024 words of electrostatic storage (ES) of 16 bits each, and operated at the then-blinding speed of 20,000 operations per second (hence its name). [WW3Q52p4] In those days we used both audio and visual aids in the debugging of programs. Originally, a scope in Test Control duplicated the X-Y deflections of the ES tubes' storage matrix so that the loop structure of programs could be seen, in action. The more often a loop body was executed, the brighter its trace. The audio aid consisted of a loudspeaker attached to a digit (I believe was digit 13 of the Accumulator) so that every time its binary state changed from a 1 to a 0 or back, a beep was heard. The execution rate of program instructions was such that loops of different lengths made tweedly tones up and down the scale, so that you could hear the various phases of the program taking place.

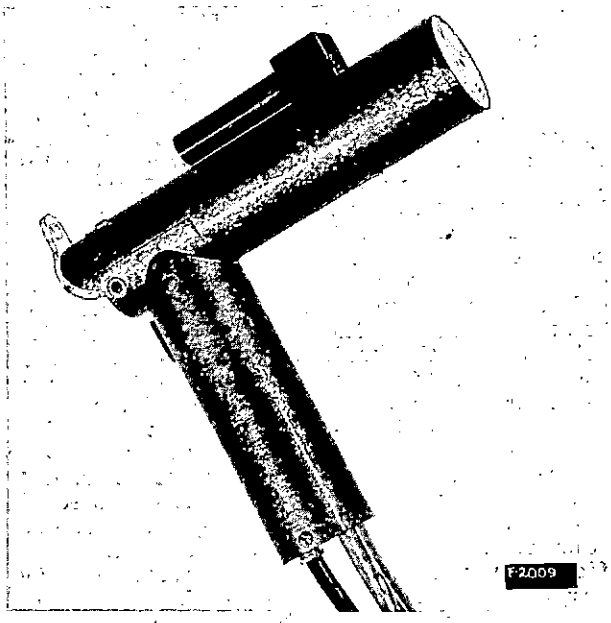
After the shutdown and changeover to the new I/O system in August, 1952, things remained stable for a time, but in the second quarter of 1953 ES was doubled to 2048 words by the addition of another bank of 17 tubes (one was a parity digit). [WW2Q53p30] Then suddenly and smoothly, in the third quarter of 1953, both ES banks were permanently replaced by the first two banks of magnetic core memory (on-line August 8 and September 5, respectively). [WW3Q53p32-34] The magnetic memory cycled at 9 microseconds versus 23 microseconds for ES, so Whirlwind now could do over 40,000 instructions per second. [WW4Q54p4] But still most loop structures remained in audio range, so the loudspeaker retained its usefulness.

The ES scope trace had been abandoned in the fourth quarter of 1952 when my first-ever program for computing the Autocorrelation function couldn't be run without causing Whirlwind to break down. Other programs sometimes gave trouble, too, but mine showed the problem to be that its tightest calculating loop fell in a group of registers which happened to fall in one corner of the square matrix of storage locations. The spherical pattern of the ES tube's holding gun (which uniformly, but with inverse square law fall-off, sprayed the entire storage surface with electrons to compensate for leakage) couldn't be adjusted for the corners without upsetting the center of the matrix, when adjacent bits were accessed so rapidly. The engineers solved the problem by changing the diode pattern of the X-Y decoders so that consecutive words were randomly scattered around the square matrix, so that even tight loops were geographically dispersed. [WW4Q52p6] My program, with a short data tape, was added to the regular maintenance test suite used for ES adjustment and checkout, and the ES memory trace scope was no longer useful for debugging, since loops didn't show clearly.

In the meantime, however, many Whirlwind programs made use of the 16 inch display scopes, and Test Control had two of them mounted high up on a rack. The lower one had a 35 mm camera under computer control, for making photographic records, and the upper one could have all 16 analog display lines ganged together to serve as a monitor check on whatever program was running. Thus the normal operational milieu for the computer operators still had the full audio/visual complex of lights, scope, and loudspeaker. When a strange sound pattern occurred, the reflex action was to glance at the monitor scope, and then start to examine the indicator lights to gauge roughly by brightness where the program was operating in memory or was the accumulator stuck in some repeating pattern, etc. That's what the operators had done, when I got that loud, panic call over the intercom.

There were more than a dozen other scopes in the cavernous black-painted, shrouded Room 222, eerily lit by dark red light. All except the Area Discriminator scope were mounted at a slant, surrounded by banks of push buttons and indicator lights in several styles of interactive consoles for the various personnel needed to perform the many functions of airplane tracking and control of the air defense mission. The most elaborate was the E31 Console of the Combat Data Director, which in addition to its elaborate set of buttons and lights, duplicated the essential master controls for Whirlwind, so that the entire complex could be controlled from that station, bypassing the normal operations of the master Test Control in the main computer room. But I, of course, had no knowledge of any of this, yet -- it was all very hush hush, and you needed special security clearance and a need to know to even take a peek.

My records don't show when it happened, but sometime in 1953 or early 1954 arrangements finally were made with Lincoln Lab to allow John Ward and me to be introduced to the enticing Room 222. I can still remember that first walk through the black-cloth-draped entryway into the red-lit, darkened room just filled with green-glowing scopes and orange-red neon indicator lights. Neat! Just what we could use to obtain mastery over the behemoth Evaluation Program, for whose complexity I already had a growing respect. The functioning of the various parts of the Cape Cod System were explained to us (although it was only later that we saw an actual test run with planes being tracked by the operators, etc.). It was clear that the E31 console, with part of an adjacent console, would be the best control station (PWS!) for general purposes, since the full complex of capabilities (including light gun and more buttons on an adjacent console) was available there along with



Final design of Cape Cod light gun, with sight. (10/2/53).

the duplicated computer controls and the intercom.

The Area Discriminator almost got lost in the shuffle. I now will give a complete description of it. All the other scopes were mounted in consoles with switches, buttons, and light gun for manual intervention. The Whirlwind light gun was shaped like a backwards pistol, with a sight close to your trigger finger knuckle and a wire coming out of a barrel which extended back over your hand. The barrel contained a photo-multipler tube, and the wire connected to an **si** line of the computer. If a displayed spot was in the sight when the trigger was pulled, that would set an activate bit (one-time, read-only) so that a suitable **si, rd** sequence following the **si, rc** that displayed the spot (but before any other spot was displayed) said that the operator had selected that spot from all the others in the display. This was used to assign tracking functions to radar blips which were returns from airplanes.

The Area Discriminator had no buttons, lights, or anything for human intervention. It simply sat in a large box in the back corner of the room. Mounted on the box was a tripod which supported a smaller box with no bottom, so that it was suspended about 18 inches over the horizontal scope face surface. This was the Light Cannon -- a photomultiplier tube mounted so that it viewed the whole scope face and could be **si, rd** read for any spot in the display. When we saw it in the Cape Cod System use, a large circular piece of dark yellow plexiglass had been laid down, centered over the scope face so that only a narrow annular ring around the edge of the scope was open to the Light Cannon's view. The yellow filtered out the blue phosphor flash of any displayed points, so the Cannon was blind to all central display activity, but any fresh radar tracks crossing the perimeter could be seen and could be brought to the attention of the Track Initiator/Monitor operator for light gun tracking. This is why it was called "the Area Discriminator", and with the plexiglass disk in place, it actually was an analog computing element component of the Cape Cod System.

2.4 Software Background

My guess is that our tour of Room 222 probably was sometime in the Spring of 1954, for the three 16" scopes mentioned in the fourth quarter 1952 Whirlwind Quarterly Report (at the time of the I/D shutdown) had grown to more than twenty, and I think all the Cape Cod consoles were complete at the time. In any case, I had my thesis to finish, Dick Turyn who had been programming on the Evaluation Program was leaving so I needed a replacement, and we needed to learn the ins and outs of Room 222 programming. Therefore we hired Bill Wolf from Lincoln Lab, solving several problems at once. He came in June 1954, but stayed only about six months, (MIT Servo Lab 1952-1958) when he left to go into consulting for himself. Later he formed Wolf Research and Development Corp., which acquired Whirlwind itself, under Navy sponsorship, when Lincoln and MIT had no further use for it. (Redmond and Smith 1980, p. 224) Wolf R&D also had the first facilities contract to run NASA Houston computers, I believe.

I guess I'd have to say that Bill was a better businessman than programmer. He did a fine job for me, writing a basic set of programs to use all the functions of the E31 console and a program to display large, fat letters on the scope, big enough to be directly read from the 35 mm film so that we could frame our classified results with "CONFIDENTIAL - SERVO LAB - CONFIDENTIAL" before and after our plots and displays. (Wolf 1954) But the programming style was very tangled and hard to follow (and I suppose not atypical of the code of other Lincoln people who were learning to master the tricks of this new manual intervention trade, for that was a major reason I hired him). Bill's code was far from debugged when he got the wanderlust. I hadn't been working closely enough with him to know what I was getting into, but I said I'd finish the debugging because that would give me the education I'd wanted anyhow. So off he went to make his fortune.

Almost immediately I saw that I would have to completely rewrite and replace Bill's work, and proceeded to do so. Many features of modularity, flexibility, robustness, and generality were needed, in order to match my growing vision of what general MIV (i.e. the PW functionality of PWS) was to be all about. Without Bill's efforts to critique, many of these aspects (which I certainly had not articulated beforehand as specifications) would not have occurred to me until much later. Also some basic principles of MIV programming (that I have seen violated repeatedly in the '70s and '80s as the microcomputer age programmers have

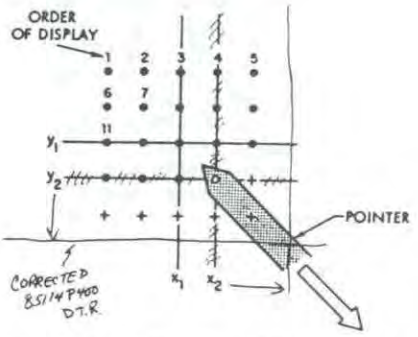
re-encountered and reinvented those 30-year old phenomena) showed up in my critique of Bill's code. For example, he had the **si, rd** to read a switch directly at the point-of-use in the program flow, throughout. I structured separate read cycles and stored the switch settings in program variables. Not only did this make the overall program better structured, but it also meant that anything that could be done manually also could be done automatically, because actual program control came from the stored variables, not from the switches themselves. This feature made possible our log and playback of manual actions as well as test simulations.

2.5 Director Tapes

It was while I was first reading, with some understanding, about the complete set of **si** capabilities of Whirlwind (including the Lincoln Lab-only portions, previously unknown to me) that I discovered that the mechanical tape reader of the Test Control Flexowriter was still operable, even though there always had been a photoelectric tape reader (PETR) since I had been around. Although the print mechanism of the Flexowriter had been in daily use all along, for operator communication, and the direct tape punch recorded some logging information, the tape reader had no current use. The number of separate tapes and the complexity of operator instructions for our Evaluation Program runs was getting so bad that I suggested to John Frankovich and Frank Helwig (on the Digital Computer Lab staff) that the Whirlwind utilities should be expanded to put all these instructions on a tape that the mechanical reader could read, while the main tapes were read in by the much faster PETR. It was worth the wait for that idea to get worked into their schedule, [WW3Q54p7] for the Director Tape program, as it was called, included not only the terse control language basics, but also conventions for labelling punched tape headers as to type (binary, program, post mortem request, etc.). When Director Tapes were further upgraded in the Summer of 1956, they still were not used by many programmers (the tape room didn't like to splice all the tapes together). [WW3Q56p23] But they were essential to our Room 222 runs, for they allowed us to completely eliminate the operators. I believe the Director Tape capability was the first true Operating System in the modern sense of the word, especially when combined with our beginning versions of Group Control (based on the earlier MDR needs) [WW4Q54p15] for manipulating drum storage. Director Tapes and Group Control are parts of the systematized solution third component of my PWS theme.

2.6 The Scope Input Program

My records don't show exactly, but I do remember dreaming up the Scope Input program on a long flight back to Boston, and my records show that I did attend an early Pre-B-58 Meeting with John Brean (the lead engineer on the Project's instrumentation work) in Dallas, on August 9, 1954, so I'd guess it was about that time. (Convair 1954 and [C54817 to S.C. Marcus, Emerson Electric]) Bill Wolf still was trying to really get his programming under way, but various test runs were being made by both him and me in Room 222. So I knew just enough to write the program. Perhaps triggered by the Dallas discussion about all the things that could go wrong with getting flight data from the plane into the computer via a crude tape system, I envisioned patching up bad spots in a plot of the input data, free hand. We already had had enough experience with the Polynomial Fit and a Lagrange Interpolation programs that had been written in preparation for the Evaluation Program, that I didn't trust any analytic methods. As the earliest written reference I have found so far (Fourth Quarter 1954) says: "A 'scope input' routine has been written which gives a type of two-dimensional analog input to the Whirlwind computer. The principle of the program is similar to the 'flying spot scanners' used on analog computers. The equipment used is a 16-inch oscilloscope under the control of the computer with a photocell mounted so that its field of view is the entire scope face. Then by programming a flying spot and asking whether or not the photocell 'saw' the spot, the program can be made to follow an opaque pointer as it is moved in a random fashion over the face of the scope. Since the program displays the spot by digital coordinates, the tracking of the pointer constitutes analog input to the computer." [WW4Q54p15]



Tracking principle of first Scope Input program August, 1954. Shadow was tracked using Area Discriminator Light Cannon. (Error correction November 1985!)

The Scope Input program is quite simple and direct. The tracking function itself consists of a tight loop that displays a 5 x 5 square array of close-spaced dots, left to right, top to bottom, interrogating the light cannon after the display of each dot. If a dot is not seen, the array is recentered on that location and the cycle is restarted from the top left corner again. Otherwise the next dot in sequence is displayed and tested. The cycle is so fast that the array can track even quite fast motions of an opaque pointer -- catching up to a new position before it can get very far.

Modified parameters of the same logic allow the pointer position to be located initially. A wide-spaced 32 x 32 array covering the entire scope face zeros in by successive

half-spacing until the tracking array can take over. Since the pointer is hand held, at least the hand will be caught by the initial spacing. This two-mode program was the full extent of the initial Scope Input program, for the intent was merely to demonstrate the tracking principle.

On that fateful afternoon, I set up the Scope Input tape in the PETR, bid the operators adieu without saying anything special, and went off to Room 222 by myself. As I approached the E31 console in the middle of the room, about to lay down my papers on its narrow desk I gasped. The restart button was there on the console, but the Area Discriminator was 12 feet away in the corner! Would I have to call for operator help after all? Suddenly it dawned on me! The top-to-bottom, left-to-right logic of the scan meant that my tracking cycle actually was a maximum-seeker-from-the-left! In other words, whatever shape made the shadow, my spot would rush up its left edge as fast as possible -- and then (by the logic) would get stuck at the very top, because it couldn't track downhill! I saw that that was why I expected it to track at all.

With that insight, my problem was solved. All of our work was done on pads of yellow paper (opaque to the Ozalid copying process). I merely tore off a blank sheet as I went over to the Area Discriminator to remove the plexiglass disk. In its place, I laid down my sheet of paper, roughly at a 45 degree angle. I went back to the E31 console and pressed the restart button. The lights indicated that Whirlwind was running. Was my program in a loop? You bet it was. As I approached the Area Discriminator I could see something was happening. There on the top corner of that piece of paper was a bright spot. The loop was my display loop. So far so good.

Very cautiously I snuck the index finger of my right hand up the yellow page, staying in its shadow. Then very gently I used my left hand to slip the paper out from underneath -- and sure enough, the spot remained -- but now stuck on my finger. I started to write. That's what got the operator's attention in the control room. The almost-random tracking recycling made a rasping sound on the loudspeaker, and that's what first got their attention. The rest is history.

I haven't found the original Scope Input program in my papers, but I do have the version used in the Symposia the following year. (Ross 1954d) (I now realize that I could have simplified the program still further, by using the left-edge-following feature to switch into tracking immediately from the first start-up dot, not seen. Then both the half-spacing logic and the initialization for full 32 x 32 spacing could have been eliminated.) Later I had our model shop make an automatic remote button-pusher, since I wasn't allowed to tamper with any of the Whirlwind wiring. A hand-held pushbutton on a long cord activated a solenoid mounted on a bracket held in place by a loosened framing screw of the E31 console frame, so that the Restart button could be pushed remotely from the Area Discriminator area. A wheeled dolly held a power supply for the solenoid, and plugged into a 110 volt outlet. It worked, and our delicate and unique relationship with Lincoln Lab regarding use of the equipment was undisturbed. All I had to negotiate was a place to park the dolly between runs!

We never got around to actually using the Scope Input program for anything, however. I didn't need it for patching our Whirlwind test data for the Evaluation Program, and for some unknown reason we didn't specify a light cannon for the Eglin Field 1103 Charactron console. We did make a light cannon for the IBM 704 and 709 computers at MIT, but again it only was used for a study of various light pen and other tracking studies in 1960, for the Computer-Aided Design Project. (Ross and Ward 1961) I never was successful in interesting anyone to make a generalized shape reader, which could easily be constructed just by changing the sequencing of the 5 x 5 tracking array (to seek minimum from the right, etc.). So in its historical setting, my Scope Input program was merely an interesting demonstration piece that the crowds loved, but was a bit before its time.

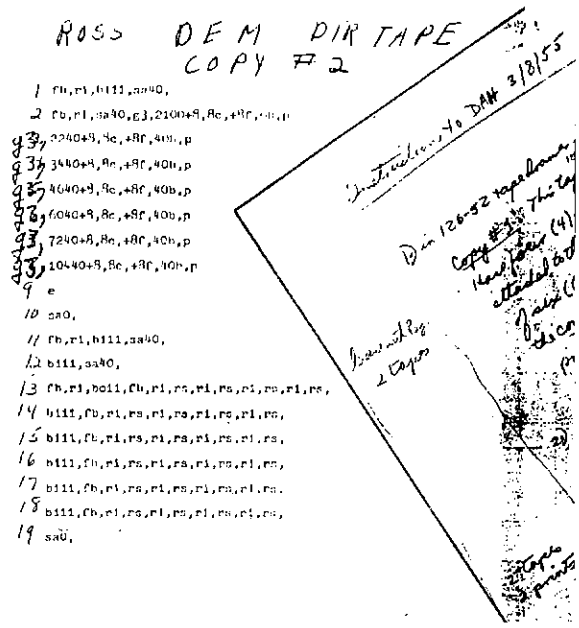
Section 3

PRACTICE

The pieces all now were in place. I could actually practice the new MIV techniques in conjunction with the Evaluation Program. I also could practice talking about what was new and exciting about this intimate coupling of man and machine.

3.1 The First Symposium (March 1955)

In any project, a point is reached when it is necessary and appropriate to demonstrate some initial results to the outside world. This point was reached for the Fire Control System Evaluation Project in the Spring of 1955. Even though my part of the project, on Data Reduction, could have used a bit more time to digest the integration of my reworking of the MIV programming with the parts of the Evaluation Program, a symposium on the Design and Evaluation of Bomber Fire-Control Systems was scheduled for March 8 and 9, 1955. (MIT Servo Lab 1955) The first day covered various engineering matters, mostly concerned with the servo dynamics of a new hydraulic antenna drive, and ended with demonstration of the drive, and (for added interest) the numerically controlled milling machine in Servo Lab. The second day covered evaluation. John Ward presented our philosophy of evaluation, John Brean the concept of digital instrumentation, and I presented the air-mass-based evaluation analysis and MIV-based data reduction programming. In the afternoon, buses shuttled two groups of about 75 people between Servo Lab and Barta Building for digital instrumentation and Room 222 demonstrations, respectively. The Visitors List shows over 170 names from 48 government, industry, and MIT organizations. It was an impressive show.



Printout of ROSS DEMO Director Tape for March 1955 Symposium. Eliminated computer operator actions and presaged Operating Systems of today.

Other than an Agenda and the Visitor's List, there seems to have been no handouts for the symposium. I have a draft copy for some glass slides, an outline of my talk, and a cardboard cutout of a fighter aircraft profile which was placed on the Area Discriminator scope, so the light cannon could talley hits on the target according to the calculated projectile position. (We used actual test data from Project Hornet, an earlier test program carried out by Emerson Electric for the B-52.)

The complexity of the Data Reduction show is indicated by my detailed instructions (dated 3/8/55 for the demo the next day!) for preparing the ROSS DEM. Director Tape. (Ross 1955a) Although I no longer can decipher the director tape language, there are well over 100 operations encoded -- loading tape segments onto the drum, executing initialization sections, calling up individual programs and executing them -- all of which would have had to be carried out flawlessly by the operator, without the director tape feature. (As a matter of fact, I had pre-loaded my paper tapes onto one of Whirlwind's magnetic tape units, for director-tape-controlled loading. The first time it worked fine, but for the second group of people the magnetic tape unit broke down! I handed the microphone to John Ward to fill the gap as best as he could, ran downstairs to the tape room files, rushed back with an armful of disjointed tapes in boxes, and with the help of the operator (probably Joe Thompson, who was the best!) managed to complete the second demo, breathlessly but in full. So we did it both ways!)

Every scope in every console in Room 222 duplicated the displays I was demonstrating at the E31 console, so everybody had a good view. On request, pertinent variables could be monitored in numerical form, the geometry of the encounter could be shown (including firing pattern, measured through the airplane cutout for the Light Cannon), and elaborate, calibrated and labelled plots of all functions could be selected. I also showed the Scope Input capability, as well, The actual demo lasted about half an hour.

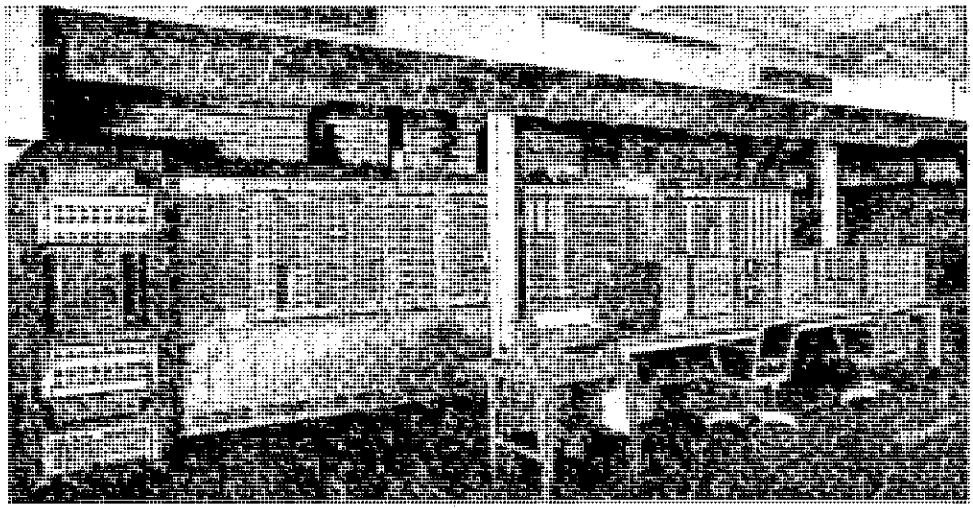
Another interesting tidbit about the demonstration is the following quote from the First Quarter 1955 Whirlwind Progress Report: [WWlQ55p56] "With the present logic the program demonstrated at the symposium had only five registers unused out of a total of about 3,500 actual program instructions. The new logic will be limited only by the drum capacity of the computer and will be more efficient as well." This documents what I remember as a previous panic point in the preparations. The final assignment of Group Control drum and core addresses came out to be almost 40 registers too big! Only by changing the words labelling the scope plots to abbreviations ("AZ" for "AZIMUTH", etc.) was I able to achieve that tight fit in the nick of time!

3.2 The MTC Charactron Demo

On April 14-15, 1955 I attended an Instrumentation Symposium at Patrick Air Force Base, Cocoa (Cape Canaveral), Florida, put on by the American Ordinance Association. (AOA 1955) The most memorable thing about that meeting was the unauthorized detour the on-base shuttle bus took to drive past a large, flat, open area with some raw concrete in the ground, with reinforcing rods jutting skyward -- all with no comment. It was the then-very-secret first launching pad for the Atlas missile, just being built.

The response to the MIV-controlled data reduction in our own March symposium had been so great that we were requested to put on another show on just that topic. On May 2, 1955 John Ward issued an invitation to a Symposium on Data Reduction to beheld at MIT, June 1 and 2, 1955. (Ward 1955) Inspired by my Cape Canaveral trip, and to provide, in addition to the Room 222 demonstrations, a demonstration of the Charactron display tube capabilities, I quickly learned how to program the Memory Test Computer (MTC) at Lincoln Lab. (Redmond and Smith 1980, p. 206) This was a general purpose vacuum tube computer that had been built specially to test the original Forrester magnetic core memory stacks. When the tests were successful and the core stacks were suddenly moved to Whirlwind, another stack was made for MTC, and it was then being used to shake down the Charactron display for use in the SAGE system.

I still have my MTC working papers for my Missile Launch Simulator program, along with the original CS Tape Preparation Requisition for "Tape 0", submitted at "4:25 PM, May 24". (Ross



Memory Test Computer (MTC) with Magnetic Core stack. (4/6/53) Used for Charactron demo in March and June 1955 Symposia on MIV.

1954a) In addition to showing examples of how the MIV monitoring displays of intermediate results would look in Charactron character display format (rather than Whirlwind's elements of a 7-stick figure eight character display [WW2Q53p31]), the program used the vector plotting capabilities of the Charactron to draw a map of the Florida coast plus some islands, and then cycled through a moving display of a missile launch, showing where the pieces would fall if the test director had to abort the mission and blow up the missile in flight. The islanders would have appreciated such thoughtfulness, I reasoned.

3.3 The Second Symposium (June 1955)

Three days later (and only one week before the symposium -- things moved fast, in those days), John Ward invited Pat Youtz (who had led all the Whirlwind work on the ES memory tubes, and now was in charge of SAGE display work) to "describe to the people at our data reduction symposium just what can be done in the near future in the way of input-output." His letter goes on:

"As you may know we are doing data reduction studies for the Air Force and have developed techniques on Whirlwind using the scope and intervention that are a great improvement over any other techniques in use at present. We are building digital instrumentation for a forthcoming field test which will be evaluated on the ERA 1103 at Eglin Field, Florida, and the primary purpose of this meeting is to get the Eglin Field people and the ERA people together and show them what we have accomplished with Bob Weiser's attachments to Whirlwind, with the hope that they will get inspired to obtain similar equipment. We would therefore like to put on as good a show as possible and that is why we would like to show the charactron tubes in operation. We have permission to use the MTC and Doug Ross has written a special display program to put the charactron through its paces.

"It seems to us that if any equipment design is undertaken by ERA that the charactron should be used as the basis for design, and we would greatly appreciate any help you or your group could give us in showing our visitors what the charactron looks like and perhaps your manufacturing and test operation. We have taken the liberty of tentatively scheduling this material on Thursday, June second, as an adjunct to the MTC demonstration at 2:00 p.m. I am sending this to you so that you can think about the business a little and I will call you Tuesday to see if we can make some final arrangements. We would be most grateful for any assistance you could give us in convincing our visitors of the current feasibility of charactron type displays. I am enclosing a brief list of the organizations who will be present at the symposium." [C55527 JEW to Youtz] Pat did indeed pitch in, and both the Charactron and the Memotron (a smaller storage tube version) were included in the June 2, Lincoln Lab visit.

On July 29, 1955 John Ward issued "Minutes of M.I.T. Data Reduction Symposium, June 1 and 2, 1955", Report No. 7138-S-1, (Ward and Ross 1955) which includes my 7138-M-112 memorandum "Special In-Out Equipment for Human Participation in High-Speed Computer Operation", July 25, 1955 (Ross 1955c) -- in which I summarized the "Program Operation" and gave a "Description of Equipment" for Activate Buttons, Intervention Switches, Indicator Lights, and Audible Alarms, each with a diagram, Description of Operation, and "Things to Notice" about how they would be used with the **EF** (external function) instruction of the ERA 1103.

For the record, John Ward's letter to Youtz shows that the following organizations

were invited to the symposium, and the minutes show how many people actually attended:

Military Physics Research Laboratory,
University of Texas (2)
Emerson Electric Company, St. Louis (3)
Air Force Armament Center,
Eglin Field, Florida (4)
Armament Laboratory,
Wright Air Development Center (4)
Engineering Research Associates,
Remington Rand, Inc., St. Paul (2)
Patrick Air Force Base,
Air Force Missile Test Center, FL (3)
Holloman Air Force Base,
Alamogorda, New Mexico (0)
Air Force Flight Test Center,
Edwards AFB, California (1)
NOTS, Inyokern (0)
Bell Telephone Labs (1)
Convair (3)
Davies (0) [manufacturer of the airborne mag tape unit]
National Security Agency (2)
Watertown Arsenal (0)
White Sands Proving Ground (1)
WCRRU, Aero. Res. Lab. WADC (2)
M.I.T., Instrumentation Lab (2)
M.I.T., Air Force Field Rep (1)
M.I.T., Digital Computer Lab (3)
M.I.T., Lincoln Lab (2)
M.I.T., Servo Lab (12)
M.I.T., Statistical Services (1)

We never had further contact with the Patrick Air Force Base people, but some years later the Sputnik era and NASA TV coverage showed elaborate Mission Control facilities in full bloom. I like to think our earlier efforts at least had some influence on those crucial developments.

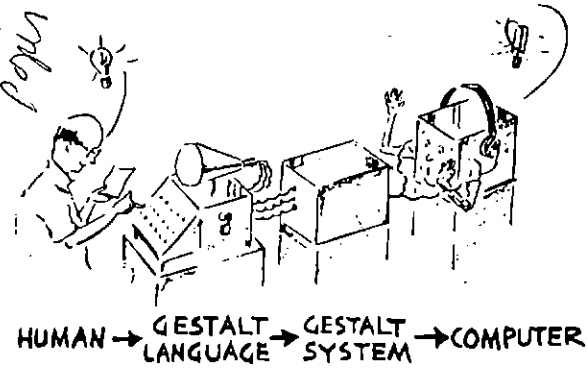
Section 4

GESTALT PROGRAMMING

The response to the two M.I.T. symposia and the clarification in my own mind that what we were building was indeed a new way to use computers prompted me to make this the topic of what would be my first professional paper. Even though all of Whirlwind may seem too huge to be a PWS, my view was and is that the PW aspect dominated the WS reality then available, so the theme of that paper and this paper is valid.

4.1 Early Views

My records don't indicate when I decided to submit a paper for the 1956 Western Joint Computer Conference (precursor to the Spring and Fall, and now the National Computer Conference series of what is now AFIPS, the American Federation of Information Processing Societies). I was Chairman of the Printing Committee for the November 1955 Eastern Joint Computer Conference, (EJCC 1955) held in Boston (we designed the conference logo, combining the logos of the IRE, AIEE, and ACM joint sponsors, which was used for several years thereafter) so I knew how the conferences ran. The earliest dated item I have (the date stamp is October 13, 1955 (Ross 1955d)) is a handwritten yellow-page draft reading "[The] Gestalt [System of] Programming -- A New Concept in Automatic Programming", with the [ ]ed words crossed out, along with an aborted opening sentence. Notice how that subtle and impulsive act makes the Title say that this is a paper about a generic concept of a type of programming. If the words were not crossed out, the paper could be about a systematic way to program -- a school of style -- or (even more specifically) the paper could be about my particular collection of programs constituting a system with that name. In the first two drafts I do use the name "Gestalt System I" in a couple of places. But when I see the cross-out now, I see it as evidence (matching the slant of the words of all drafts of the text) that I always had the generic concept in mind as the subject of the paper.



Communication from Human to Computer

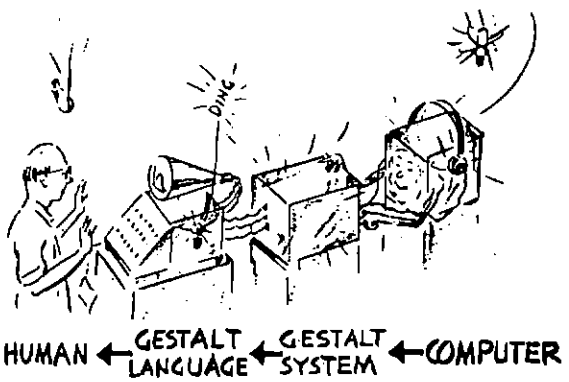
I remember that from the first rush of insight that so entranced me in that first exciting visit to Room 222 I knew that things would be profoundly different when man and machine were intimately coupled. I always had a broader view than just our data reduction application in mind. The earliest record I find on our use of MIV is in the Third Quarter 1954 Whirlwind report, which is suitably subdued, but also clearly is targeted to experimental programming:

"Work is progressing on another phase of this problem, using auxiliary in-out equipment both as an integral part of the data-reduction programs for monitoring purposes and as an aid to experimenting with improved computational techniques. Intervention registers and scopes will constitute the working parts of this system with efforts being made to provide the flexibility normally associated with analog computers. Present work is concerned with the development of routines to decode intervention-

register inputs and to utilize various specialized scope outputs. After merging these routines with the data-reduction programs, an elaborate logging scheme, using magnetic-tape output, will be written for permanent records of all operations and interventions during an experimental run." [WW3Q54p15].

But in an August 11, 1955 letter to my Oberlin classmate, Ben Scheff, urging him to come and work with me at M.I.T. following his Navy tour at the National Security Agency, more of my real feelings show:

"The work on fire-control system evaluation has been extremely interesting and has grown continually in scope and capabilities. Now we are really on the forefront of a whole new way of using large digital computers. We are in the fortunate position of being purely a research group and never have to get our hands dirty with actual data reduction. Once we get a new technique developed and proved, we drop it or expand on it to new, more comprehensive work.

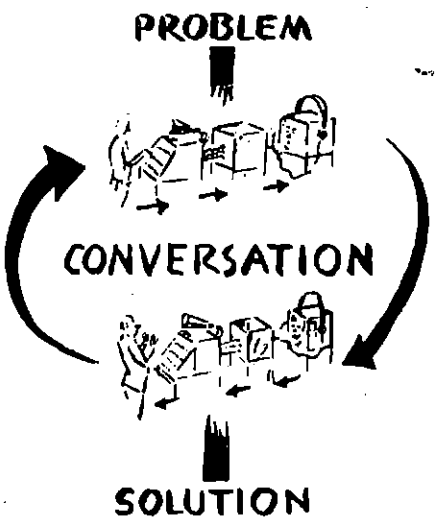


Communication from Computer to Human

"A large part of our efforts go [sic] into developing new techniques and programs to achieve more flexibility in devision [sic] and testing new programs. The MDR report, which is enclosed, is one such program, and is a pretty good example of the kind of programming and philosophy we like to follow. The other special report on our Symposium gives some idea of the techniques we're now exploring. Both reports are now out of date but give the ideas.

"It is really amazing the complexity which arises in making the computer act in these ways, but will be a really nice system when it's done. We have an intermediate version running now which we demonstrated at the Symposium, but the newest version (almost complete) will run rings around it. The new system is really much more general than we now require but it will essentially allow us to program with programs while the computer is running instead of coding for weeks before a run.

"In a nutshell, what we're doing is fixing up Whirlwind so that we can manipulate its operation while it's running, turning on and off programs, examining and modifying so that we can use the computer as a research tool on an idea or concept level, rather than on the grungy coding level. Frankly, it's pretty exciting work ... I can honestly say that I don't know of any other work in computers and their use which is more interesting than what's going on here." [C55811]



Solution by Conversation

4.2 Initial Drafts

I have several distinct drafts of my attempts to write the Gestalt paper. The November 17 version actually submitted to Byron Bennett (SRI), WJCC Program Chairman [C551117] is completely different from the October 13 draft, and that paper, in turn, was completely reworked several more times before the final submission of our Project Technical Memorandum version (Ross 1956c) on February 1, 1956, for publication in the Proceedings. I think it is historically important to dwell briefly on the evolution of my expression of my then year-old, but still elusive ideas, because such an analysis will show my present PW versus WS theme in the struggles of its original context.

The original (October 13) draft jumps right in with the dictionary definition of Gestalt psychology: "The theory that ... events do not occur through the summation of separate elements, ... but through formed patterns of these, integrated units which function singly or in interrelation," and proceeds to say that standard programming practice is to be replaced

"by simple and unambiguous expression of the desired characteristics of behavior, the expression itself in effect tying together integrated units of computer behavior ..." (presaging today's declarative languages, functional programming, and the reusable components of software engineering). And still in the opening paragraph, "automatic coding systems are aimed at easy communication between the human and the computer, whereas a Gestalt system of automatic programming is aimed at easy conversation ..." [a phrasing with which I was particularly pleased, and which I retained in all further drafts]. (Ross 1955d) These are indeed the pertinent ideas, but the later drafts are much more roundabout in approach. At the time, even ordinary programming was still immature, and there was a complete lack of the common, shared culture we know today. The ideas were very hard to formulate and very hard to assimilate.

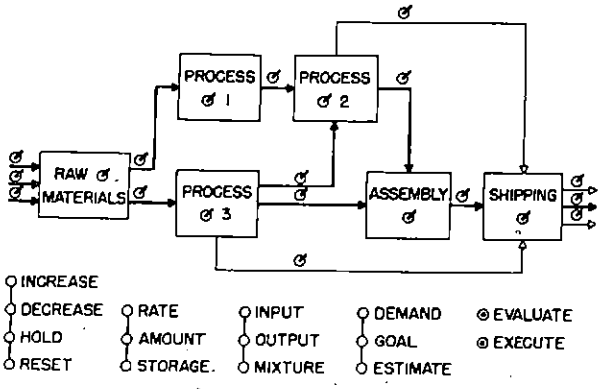
The second (November 17) draft (the basis for acceptance of the paper) begins with the following "Introduction:

"In any human endeavor there are three major phases: conception, expression, and execution. Gestalt programming is an attempt to make these three phases as nearly identical to each other as possible with respect to computer programming. In this paper the word Gestalt is used to mean a concept of a task to be performed by a computer. In a Gestalt system of programming the Gestalt, or idea, is expressed simply and unambiguously in a special language, rather than through the laborious assembling of machine-codes, pseudo-codes, subroutines, etc. Using a Gestalt system, the expression itself in effect ties together integrated units of computer behavior, which function singly or in interrelation, to achieve the desired effect. The purpose of a Gestalt System is to facilitate the transmission of general ideas as in a conversation, between a human and a computer, so that the maximum use of their respective capabilities can be made.

"After presenting the abstract theory of Gestalt Programming this paper discusses several Gestalt Systems in use today at the Massachusetts Institute of Technology and describes briefly the types of computer hardware which are best suited to this application." (Ross 1955f)

This was retained as the Synopsis for the final paper, but the body was reworked many times. The referenced MIT systems were the Whirlwind Comprehensive System (CS) itself (including mention of Director Tapes), the Whirlwind Marginal Checking facility (which required a technician to classify oscilloscope traces, etc., through MIV, and then isolated faulty tubes and components), and our own Data Reduction and Experimental Programming System.

In addition to various attempts to describe problems in which MIV was desirable (not an easy task in the hands-off, standard mode of submitting runs to a computer operator) I needed to have an example to illustrate the actual design and programming steps. My draft pages show one attempt using the LIP and RIP (Left Insertion Panel and Right Insertion Panel) and various buttons for CS computer operations, but that was not used. Another complete development says "Consider the launching of the man-made satellite schedule for the coming Geophysical Year," including the suggestion that a "part of the Gestalt Language might be a kind of joy-stick, so that the human could steer the rocket remotely." (Ross 1955e) But I decided that this example would seem too flashy, and instead worked out an example of Automatic Factory Control for the final paper (perhaps a prescient choice, since my soon-to-arrive role in APT was unknown to me then).



Gestalt Language for Automatic Factory

4.3 Oral Presentation (February 1956)

Then there follow endless yellow pages of my attempts to work out an oral presentation of the paper (Ross 1956a) ("The title of this paper has probably raised questions in your minds both about what I am going to say and why I have chosen to introduce a new word-usage into the already crowded computer terminology" -- scrubbed, fortunately!) I presented an interdepartmental colloquium version at MIT entitled "Talking with Whirlwind -- An Introduction to Gestalt Programming" on January 11, 1956 (rescheduled from December 14), [C56016 to Bennett, and (Ross 1955f Addendum)] and finally managed to do a decent job as the opening paper at the San Francisco WJCC on February 7. Electronic Design magazine enthused:

"Programming and coding sessions at the Western Joint Computer Conference, San Francisco, February 7, 1956, were attended by overflow crowds. Highly significant papers were read. Here is a verbatim report of the impressions on [sic] the session chairman, Francis V. Wagner, group leader, engineering computing, North American Aviation, Inc.

"The paper on 'Gestalt Programming' by Ross, which opened the session, was a milestone of forward thinking. It refused to accept the common fallacy that human beings can contribute little when teamed directly with automatic

digital computers, and took the position that only the more versatile human mind can, at present, handle some parts of some problems. In the past, many people have noted that, to exploit such a partnership, there is required not merely a satisfactory language for communication from the human to the computer but a language which allows fluent conversation between the human and the computer., Mr. Ross went beyond merely stating this, [I thought it was original with me, at the time! -- D.T.R.] and presented evidence of some serious thinking towards analyzing how such a language ought to be designed. The principles which he stated and the illustrations which he gave of ways to design such a language, and of the hardware which might form the medium for the conversation should go far towards clarifying the problem for future implementation of these ideas ...

"Audience participation was distinguished more by quality than quantity. No more than three of four questions were asked of each speaker. [Wagner's overall impression of audience was that on the average they probably were not prepared to comprehend the concepts presented. -- E.D. Editor] Although these were put by highly competent people, and indicated careful evaluation of the paper, they were aimed more at underlining or amplifying certain points. In no case did they take violent issue with any of the statements or implications of the speaker, nor did they open up any controversial issues." (Electronic Design 1956)

Even the Scientific American expressed an interest in the paper. A suitable summary of my thinking at the time is the following quote from my January 12, 1956 reply to Dennis Flannagan, Editor, enclosing a copy of the paper. Actually I sent the final, TM version -- but I still was concerned about its adequacy in expressing my ideas. This probably was my own dissatisfaction with my MIT talk the day before, even though it was very well received. A few Lincoln people let me know by body-english that they thought I was plagiarizing other people's ideas, however. It was clear that they thought the ideas were important -- important enough to let me know their feelings -- but I, in turn, saw their upset as evidence that they really didn't see the differences between the specific Cape Cod incorporation of the man in the loop as an engineering necessity, any my generic Gestalt Programming idea. Somehow I wasn't expressing myself clearly enough, or they wouldn't feel that way, I thought. Hence I wrote to Flannagan:

"Since it is not stated too clearly in the present draft, let me try to summarize for you here, what Gestalt Programming is and where it fits into the established computer technology. As computer techniques have developed over the last few years, there has been a growing trend toward more sophisticated methods for connecting the human, who states the problem, to the computer, which is to solve the problem. Out of this trend has come, as a natural consequence of the maturing technology, a desire to use computers for solving problems which cannot be completely specified in terms which the computer can handle. These major trends are united into the general problem of using humans and computers together to solve problems. An analysis of this problem shows that more efficient and natural languages are required to bridge between human and computer, and that these languages must operate at the idea or concept level. In particular, a statement in the language must lead directly to the solution, in the same way that a Gestalt (in the psychological sense) unites at once, basic units into a single entity or pattern. The theory of Gestalt Programming is an attempt to set the initial outlines of the structure of these languages, and to indicate how to construct and use them. The physical content of Gestalt Programming is not new, but the emphasis and point of view is new, and is intended to clarify the directions in which these fascinating developments can go." [C560112]

But nothing further came from the Scientific American interest, which is too bad, because unbeknownst to me, at about the same time (and completely independently) George Price was in communication with the editors of Fortune magazine, preparing his landmark article on The Design Machine, which appeared in the November 1956 issue. Francis Bello, Science editor for Fortune wrote to me on March 27, 1957 that

"Price wrote us in November, 1955, suggesting his article and we had a rough draft in our office on January 15, 1956, which outlined the major features of his Design Machine. I mention this because it may surprise you that the article was so long in preparation before appearing in Fortune. (And certainly you would be entitled to wonder if Price had heard of your February paper.)

"What surprises me still more is that computer experts at I.B.M., Sperry Rand, and Lockheed, who read Price's paper for us, and checked his computations, had not heard of -- or at least gave no indication that they had heard of -- your work in this area.

"From the beginning our concern was that Price's ideas were too blue skyish for serious presentation. The world keeps moving faster than Fortune can keep up." [C57327]

To which the only comment (valid today, as it was then) is -- it does indeed. Before I address Price's article, there are more items from our own work to be covered.

Section 5

CONSOLE DEVELOPMENTS

With the PW component established, next comes the WS component of my PWS theme. I can cover here only the earliest developments of what was to be a long term evolution of computer graphics and keyboard language techniques, both hardware and software, well into the 1960s, at MIT.

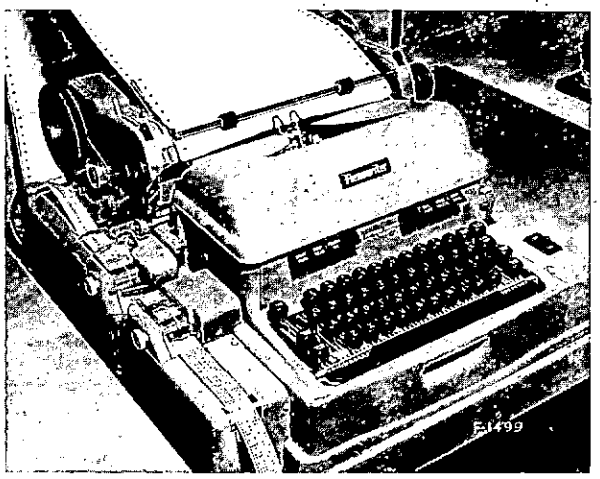
5.1 Direct Keyboard Inputs

It's funny how memory plays tricks on us. For years I have known that it was at my instigation that direct keyboard input to Whirlwind was installed (beginning the chain of events that eventually led to the Compatible Time Sharing System (CTSS) and Project MAC at MIT in the 1960s). But at the same time, I thought that all of our MIV and Charactron console design and installation for the Eglin Field 1103 had come after and had been based upon what we already had going on Whirlwind. The records show that I was mistaken. The Whirlwind keyboard proposal actually followed my proposal for the 1103 keyboard. That came as a surprise to me, when writing this paper.

The earliest date I find in my own files regarding the ERA 1103 is a collection of general information (brochure description and Installation Instructions -- where to install the motor generators and air conditioning, etc.) bearing date stamps of John Ward and myself of November 8, 1954. (ERA 1954a and b) Two yellow pages of notes about the order code indicate that this was my first introduction to what was in store. The earliest Charactron information is a collection of Convair Charactron Project documents on the Model 708 Charactron Display Console, the Model 100 Computer Readout System, and the Rapromatic Camera and projection system, all with my date stamp of October 18, 1955. (Charactron Project 1955)

By December 14, 1955 I had written a memo on "Proposed Equipment Modifications for the Pre-8-58 Data Reduction Programs". (Ross 1955g) It recommends that AFAC should procure an additional 4096 registers of magnetic core memory beyond the 1024 electrostatic storage registers that were standard for the 1103, and "that the photoelectric tape reader and the Flexowriter connections be modified", in addition to installation of the "Type 70-A Charactron with photographic facilities under computer control, reported already on order", as well as "the Intervention Switch facilities, reported already on order."

Engineering Research Associates (ERA) definitely was a company of (outstanding) engineers -- not programmers. I believe the AFAC 1103 was one of two "Serial One" first models of the ERA 1103, (Cohen 1985) and I encountered several features that were good engineering ideas, executed in such a fashion as to make programming and use most awkward. The 1103 had a 36 bit word length, and an "Arithmetic Section" with Accumulator (A of 72 bits), Shifting Register (Q of 36 bits), Exchange Register (X of 36 bits), and two Input-Output Registers (IOA of 8 bits and IOB of 36 bits). The binary form of 7-channel punched paper tape, called "bi-octal", was prepared off-line using a box full of relays with a 10 character per second mechanical tape reader on one side and a mechanical tape punch on the other. A number of empty sockets into which jumper-wired plugs could be inserted allowed the



Flexowriter with 10cps reader (front) and punch (back). Keyboard was not used for Whirlwind direct input until summer of 1956, which led to later timesharing projects at MIT.

user to control the tape transformation between the two. Absolute address assembly language tapes punched on an off-line Flexowriter could be converted to bi-octal with one set of plugs.

We contracted with the MIT Digital Computer Lab to have a WWI-1103 Input Translation program, based on CS, built so that we could write 1103 programs with "flads" (floating addresses) etc., translate them on Whirlwind, fly them to Florida, and debug and use them there. [WW1Q55p56and79-81] The 1103 itself had no such software. I wrote a companion 1103 program, handling only absolute addresses, to support our on-site debugging efforts. (Ross 1955b)

For this early machine, things were pretty awkward, the ERA photoelectric tape reader (PETR) could only be started manually, the PETR was connected directly to the Q register, thus entangling the Arithmetic Section unnecessarily, and worst of all, as my memo says:

"The bi-octal tape format now in use does not allow the use of programmed checks to insure that the information was read from the tape correctly. Since the read-in program for reading bi-octal tapes is now wired into the computer, the increased flexibility required to change tape format is not presently available.

"There are two major approaches to the problem of placing the photoelectric reader under computer control. A Ferranti tape reader could be purchased and installed exactly as in other 1103 computers. This is the best, but most expensive solution. The other possibility is to put the drive motor of the present ERA tape reader under computer control, and to have the photocells read into IOA (or IOB) rather than into the Q register. The only disadvantage to this inexpensive method is that a few inches

of extra tape feed-out must be placed in the tape wherever it is to be stopped, since the ERA reader is not a fast start-stop device like the Ferranti reader. However, this method was used successfully with a similar ERA reader for several years on the Whirlwind computer at MIT, before the Ferranti readers were obtained." [p. 2] (The inexpensive method was chosen, and the WWI-1103 Input Translation program was forced to ensure the proper blank tape sections were present.)

The memo then continues with similar observations regarding the keyboard:

"6. Flexowriter Input

"To augment the Manual Intervention facilities mentioned above, it is suggested that the Flexowriter presently attached to the AFAC 1103 computer be connected to act as both a keyboard input and mechanical tape reader input device. This method is considered preferable to the purchase of special keyboard input devices at this time because, again, it should be very inexpensive, and will give considerably more flexibility than any commercially available keyboard of another type.

"An essential part of Manual Intervention Programming is that decimal (or octal, or English) information can be communicated directly to the computer. In use, this system would use intervention switch settings to tell the program how to interpret the Flexowriter keyboard input at any given time.

"The mechanical tape reader probably could use the same circuitry as the keyboard and would be used to read control tapes (while data or program tapes are in the photoelectric tape reader) so that manual control will not be required for routine operation of the computer system, which will result in greater reliability.

"A comment contained in 1103 Newflash No. 15, December 7, 1955, seems to imply that the keyboard of the Flexowriter on the Ramo-Woolridge 1103 is already connected in this way, so that the required engineering may be already available. The comment does seem to imply, however, that the 'directly connected typewriter' uses the Q register. It is much better to use IOA (or IOB) for this purpose so that normal operation of the arithmetic element is not impaired. It is very possible that both the keyboard and mechanical reader could share a major portion, at least, of the circuitry connecting the photoelectric tape reader to IOA (or IOB), with resulting savings in engineering and hardware." [p. 3]

It is interesting to me that even though I chide the engineers for their lack of vision, I suffered similar myopia regarding how to establish controls. I'm sure that it would not have taken significantly more programming to have an opening character or so to specify what type of information was on the tape, instead of using MIV switch settings. This had, in fact, already been done for the logging of CS tapes on Whirlwind, after all, when Director Tapes were introduced. Our "modern" views take time to evolve and develop, I guess.

In any case, this December 1955 memo shows that the AFAC 1103 keyboard suggestion formally came before the Whirlwind keyboard suggestion, and in a fashion that surprised me a second time, in my researches for this paper. I had found in my files my memo 7138-M-144 "Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer", dated March 1, 1956, (Ross 1956c) and my memo 7138-M-148 "Flexowriter Keyboard Input to WWI, Preliminary Specification", dated March 14, 1956. (Ross 1956d) But digging still deeper I found that my memo 7138-M-160 "A Proposed Flexowriter Input for the Whirlwind I Computer", dated April 30, 1956, (Ross 1956f) was the formal proposal finally sent on to Lincoln Labs for formal consideration, with the concurrence of the Scientific and Engineering Computations Group and "several groups in Lincoln". So at this point in my sleuthing, I was convinced that the actual sequence for my on-line keyboard suggestion definitely was first 1103, then Whirlwind.

The second surprise came from a detailed reading of collected draft papers for a May 8, 1956 revision (to "144A") of my 7138-M-144 March 1 memo, this time done by John Ward, (Ward and Ross 1956) for formal forwarding to AFAC and our WAOC sponsors. In that revision, which includes John's redesigns of Whirlwind drawings for Intervention, Activate, and Audible Alarm circuits, is a large fold-out drawing entitled "Keyboard Read In System, WWI", showing Lou Norcott, "Eng." and Larry Holmes, "Appd.", both dated "2-27-56" -- with the hand-scribbled notation "see him [Norcott] or Doug R. O.K.". The interesting surprise to me is that this shows that I already had been active during February, and probably before, working behind the scenes with the Whirlwind engineers to confirm feasibility and costs before sticking my neck out with formal suggestions. Therefore the actual fact is that I can't sort out which came first, my thoughts regarding the 1103 or Whirlwind. My guess is that they surfaced almost together. I.e. I knew general keyboard input was needed for Gestalt programming, but the fact that the ERA 1103 already had an on-line keyboard of some sort triggered me into action. I first worked the idea out on the basis of Whirlwind, working with Larry Holmes, but the first actual writing was about the 1103.

Larry and I played the game to the hilt, for I also have a short interoffice memo from him to me, dated April 3, 1956 (Holmes 1956) which begins:

"I received a copy of your Memorandum 7138-M-148. The attached chart illustrates the five different plans that you might consider in the determination of a decision as to which installation you will request." An then goes on to conclude:

"I sense that it would probably be the middle of June before the chosen facility would be operational.

"If any additional information is required for your subsequent memorandum to management, I will be pleased to procure same."

Bob: place this remin history with i/o devices SW interaction

5.2 Microprogramming for Whirlwind

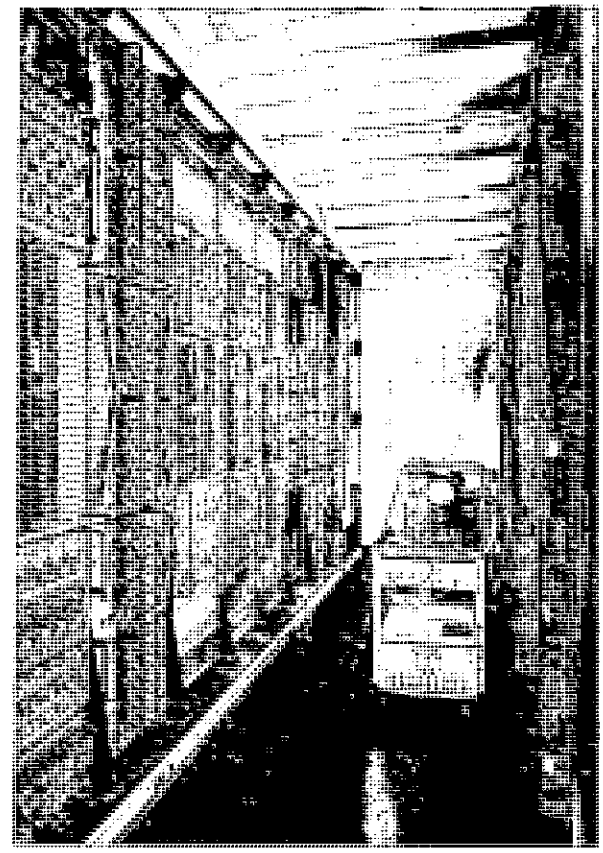
It was not all that unusual that I was in close touch with the Whirlwind engineers, for another one of my on-the-side, recreational projects at that time concerned a plan I had to modify the internal logic of Whirlwind I to allow experimentation with microprogramming. This idea had come to me in the same spurt of inspiration when I studied the Lincoln-only, si-based input-output lines that led to the original Director Tape proposal. I discovered that not only was the mechanical tape reader still available (as I have already covered), but much more importantly that there was a whole additional flip-flop register, called PR -- the program register -- just sitting there, completely unused! The PR had temporarily held the memory address for Electrostatic Storage operating and was no longer required for magnetic core memory. It still had all the wiring that connected it to the WWI Bus, and had three outgates per digit! What a treasure trove! In those days of discrete components and tubes, flip-flops and gates were very expensive (several hundred dollars per bit), and here was all that gear, cut out of use by the modifications to the main machine, but all ready to be fixed up if I could find a use for it.

Of the papers I still have, the earliest date stamp I find is that of "Engineering File #2, August 18, 1955" on copies of large drawings of "Timing Diagrams, WWI", Digital Computer Lab drawing E-51744, and "Control Pulse Output Connections, WWI"; DCL drawing D-531?? (torn off), which were made for my use. My own vellum master drawing of "Proposed μ-Operation Schematic for mi Instruction, WWI" is dated April 5, 1956, and numerous copies show my steps of refining my design after that date. (Ross 1956e)

On March 1 and 2, 1956 a seminar on Microprogramming, sponsored by the MIT Digital Computer Laboratory was held with papers listed by H.R.J. Grosch (GE), O.J. Wheeler (Cambridge U), N.R. Scott (U of Mich.), L.C. Hubbard (IBM), B.D. Smith (ERA), J.J. Eachus (Datamatic), H.P. Donahue (NSA), G.M. Hopper (Remington Rand), J.W. Wegstein (NBS), and J.W. Carr (U of Mich.). (MIT Dig. Comp. Lab. 1956) I was not even on the list, because the Digital Computer Lab organizers didn't know about the hobby project I'd been doing only by myself, with a few interactions with the Lincoln Lab Whirlwind engineers. I hadn't even gotten an invitation to the seminar, but when I heard about it, I was welcomed, and went.

With a June 6, 1956 date stamp, I got pages of a typed transcript of the seminar from Frank Helwig which shows that at the last minute I had been squeezed into the program before Wheeler's second talk. As is usual for my oral speaking, the transcript is almost incomprehensible, but so is the extended ensuing discussion by others.

I did manage to complete the design of the proposed mi microprogramming instruction and



Control Matrix of Whirlwind. Wires, diodes, and gate boards laid out translation of 5-bit instruction code into 120 Command Pulse Output (CPO) micro operations.

my papers do include a complete outline of the report I intended to write, but never completed. (A "to-do" list, buried in these papers starts with "Flexo keyboard memo - Fri; AFAC int sw. diagram; Pre-8-58 MIT meeting; Gestalt Paper to AFAC, etc.; mi-WWI memo" and a slew of other items, including "Numerical Control Personnel", so I'd guess the date was around May, 1956 when the outline was done.

The idea of the proposal was as follows: When you walked inside the bowels of Whirlwind, down one aisle was a mass of bare copper wires and boards of components, neatly arranged in a matrix. First came five bare vertical wires, and then 32 long, horizontal, bare wires going down the length of the aisle. These, in turn, overlaid 120 vertical bare copper wires, each of which terminated incircuitry boards arranged along the bottom, by the floor. There also were eight coaxial cables running horizontally, with one or another feeding each of the lower circuitry boards. These cables carried one of eight time pulses, and the original five vertical wires supplied the 5-bit instruction code of the instruction now being executed. This was the Control Matrix of Whirlwind. The neat thing was that soldered right there in front of you, at the various crossing points of the five vertical wires and the 32 horizontal

"You understood the whole computer just looking at it."

wires, was a collection of germanium crystal diodes that spelled out the binary number system, so that each of the horizontal bare wires represented one of the 32 instructions of Whirlwind. To understand what each instruction did, you had merely to similarly read the pattern of diodes soldered between its horizontal wire and the 120 vertical wires. The 120 vertical wires each selected one of the 120 CPO Units (Command Pulse Output gates) of Whirlwind, each of which could do some micro operation (such as clear the I/O Register, read from memory into the A Register, etc.) when its corresponding time pulse arrived. In a nutshell, my plan was to use the flip-flops and gate connections of the PR to dynamically compose instruction steps from a selection of the existing CPOs -- as though a new horizontal line had variable diode connections to the CPOs.

The report outline indicates that I found a subset of 39 CPO micro operations that would be useful in various combinations. My design allowed any number of them to be chained together on time pulses 6, 7, 8, and 1, interspersed with memory addresses and I/O operations. Thus microcoded instructions of any length and complexity could be composed and executed, interspersed with standard Whirlwind instructions. As the outline says "Very fertile field for problems which can use micro-program in MIV Gestalt Programming." The report was never written, however, and the change to Whirlwind was never actually proposed (I had in mind a big gang switch that would include or exclude the mi instruction in Whirlwind's repertoire) because when I programmed various test cases with microprogramming and then with ordinary Whirlwind programming, there was very little to be gained. The trouble was that Whirlwind was so closely designed (to save expensive gates, etc.) that the A Register formed an insurmountable bottleneck. As the outline says ("md" is "multiply digits" -- a masking operation):

"Fundamental difficulty with present WW construction for this type of application since only entry to AC is thru AR so that usually AR is not indep. of AC. Also this necessitates use of more steps than really required since first must load AR before can work on AC. Also would be good to have a working operation built into circuitry to give effect of md's since in logical ops this is often fundamental but is lengthy μ-op-wise or md-wise since another mi is necessary to get back to μ-mode.

"Conclusion:

"Study of this report and independent investigation of this kind of approach to μ-prog is very instructive on problems involved. If more major changes in WW structure were acceptable might be able to obtain a useful gadget. Best bet, however, is to contrive basic idea of investigating μ-computers by similation without changing WW."

So my grand scheme for microprogrammed MIV and Gestalt Programming came to naught.

5.3 Keyboard Input for Whirlwind

The direct Flexowriter input for Whirlwind was built, however, and could be operated from either the E31 Console or from Test Control, as I had suggested. The Second Quarter 1956 Whirlwind Progress Reports says:

"Since 1954, the MIT Servomechanisms Laboratory has been using the WWI manual intervention and display equipment in the development of high-speed data reduction techniques. In order for them to expand their research into computer applications, it was essential that more versatile manual inputs be made available on the WWI computer. Besides requiring additional on-off switches, many of the new programs will be so complex and will require so many parameters that the only reliable way to instruct them will be to use specially designed mnemonic languages and translation programs. In order to have this general language structure available on a manual intervention basis, it is necessary to have a keyboard such as a Flexowriter for direct input to the computer.

"The MIT Scientific and Engineering Computations Group have contemplated the following applications for the new facility:

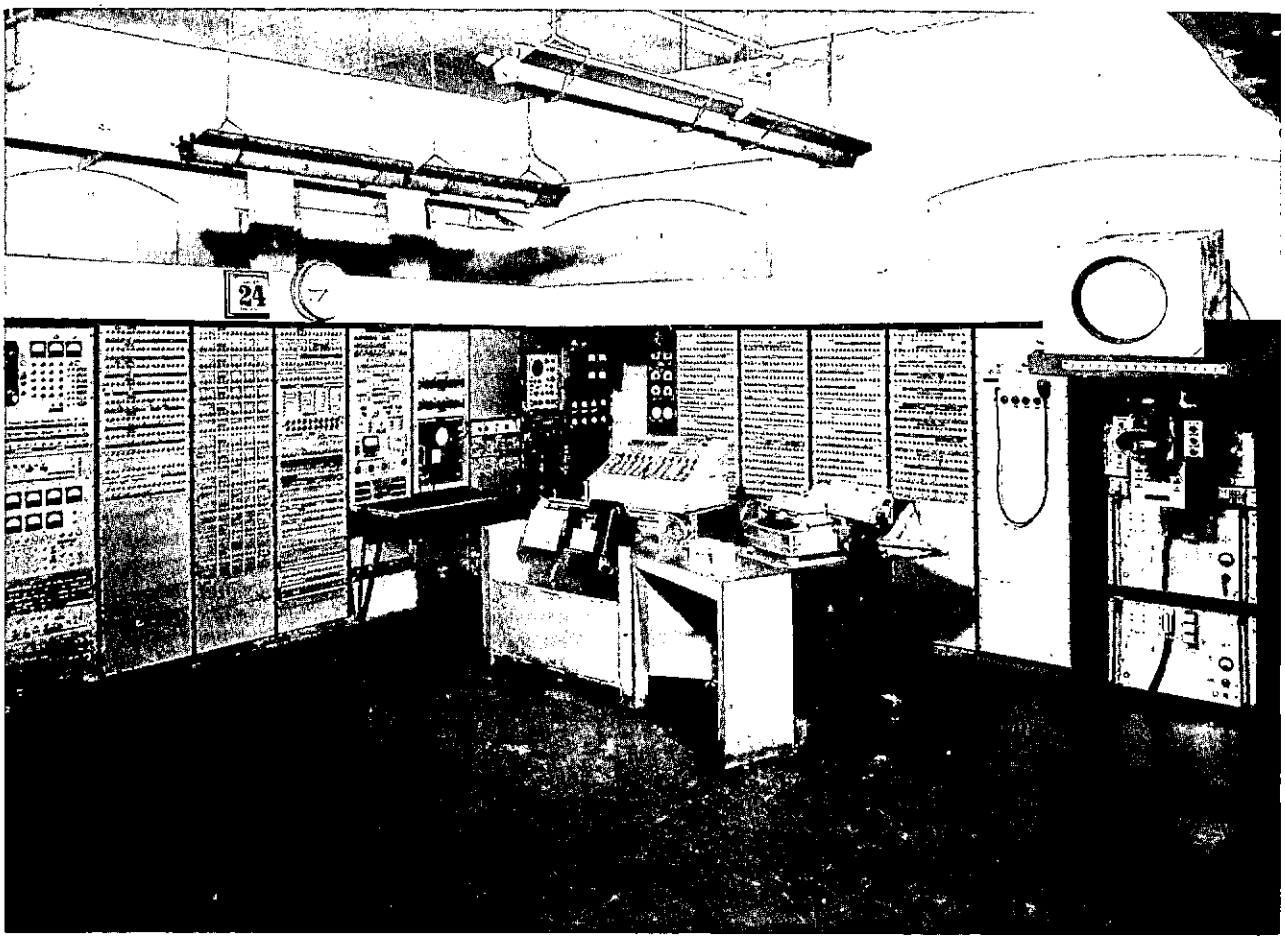
"1. Demonstration programs would be a great deal more effective if this form of input were available for control purposes.

"2. Typewriter input for Comprehensive System Flexowriter and post-mortem request tapes. Short program modifications and post-mortem requests can presently be inserted in the insertion registers. However, errors are easily made because the required vocabulary is awkward. A typewriter input facility would make available a normal mnemonic vocabulary for such purposes.

"3. Experimental use of a typewriter facility for direct operator control of the computer. Here we would consider using the typewriter to replace the button-pushings required of the operator during normal operations. Vocabulary similar to that of director tapes and performance requests would be devised for these purposes. This could easily prove to be an extremely convenient and efficient method of computer operation.

"The new input installation will be available for use by 4 July 1956. Much of the information to be inserted via the keyboard will be the same as is now introduced via a free running photoelectric tape reader using punched paper tapes. The keyboard input will also be treated as a free running device, i.e., selection of the facility by the computer may be followed by an arbitrary number of read instructions, each of which reads the next character which has been struck on the keyboard. The total equipment requirements amount to 15 relays and 20 tubes." [WW2056p63]
My original M-148 proposal specified that

"Since much of the information to be inserted via keyboard will be the same as is now on paper tape, it would be best to have the keyboard operate in a free running mode, i.e. an si selecting the keyboard may be followed by an



Whirlwind Test Control Console (1/2/57) showing MIV panels for computer operation behind Photoelectric tape reader (PETR) and Flexowriter. Monitor (above) and camera (below) output scopes are on right.

arbitrary number of rd's, each of which reads the next character which has been struck on the keyboard. In this way, programs such as the entire CS system can be modified to accept keyboard rather than tape input merely by changing one or two si instructions." (Ross 1956d, p. 2]

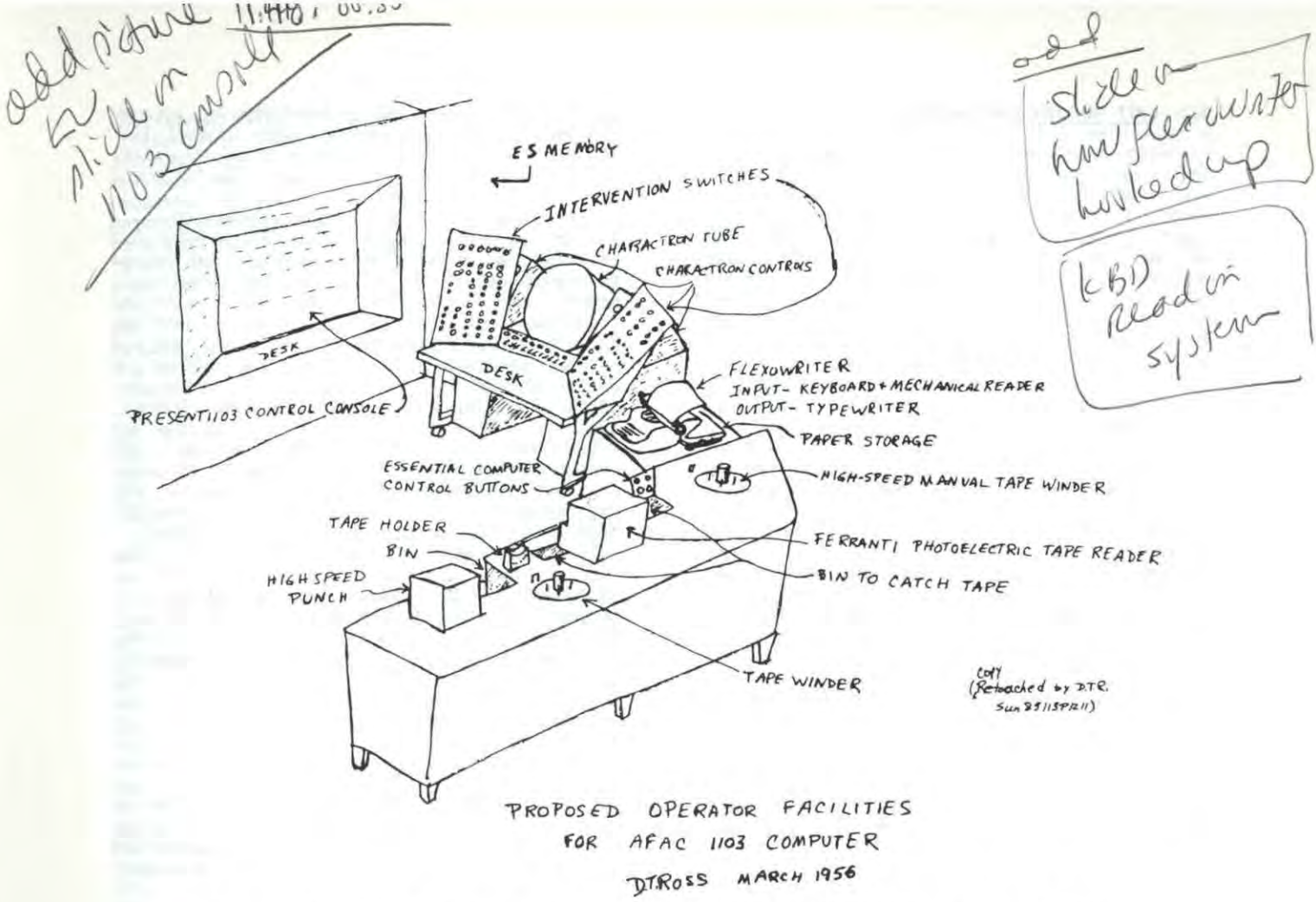
The existing Test Console Flexowriter was selected by one si, and another Flexowriter on a wheeled table plugged into the Room 222 E31 Console and was selected by a different si. We used it some with our SLURP developments, but Lincoln people didn't take advantage of it, to my knowledge.

While digging through my papers, much to my surprise I ran across a Oivision 2 Lincoln Lab memorandum dated February 2, 1959 (three years later) on "The Oirect Flexowriter Keyboard Input System at Whirlwind I (Barta)" by C.F. Brackett in which I discovered that

"A pushbutton on the direct output Flexo table [in Test Control? - O.T.R.] allows the Flexo keyboard to be used as a direct input device. If this pushbutton is held depressed and the Flexo keyboard operated, the Flexo code for that character will be set up in the keyboard Input Relay Register where it will be available for later transfer to the In Out Register ... Since the keyboard Relay Register is cleared of old information during the read in of new information, the operator should normally wait until the indicators are extinguished before typing the next character." I never knew it had such a button. Maybe it was only on the Test Console Flexowriter. If the Lincoln memo indicates what actually happened when my idea was passed through the Lincoln approval cycle (I never had occasion to use it myself, as I remember), no wonder that first installation was little used. The button would force pick and peck typing! This equipment did, however, serve as the basis for later direct keyboard inputs to the IBM 704, 709, and then 7090 computers at MIT, which led to the initiation of time sharing.

5.4 The Charactron/MIV Console for the AFAC 1103

The Charactron/MIV console for the Eglin Field AFAC 1103 as finally proposed had two 36 bit Intervention Registers, one 8 bit Activate



Sketch for Charactron/MIV Console for ERA 1103 at AFAC, Eglin Air Force Base. Built by Stromberg Carlson based on Whirlwind experiences of Servo Lab Data Reduction Project.

Register, one 36 bit Indicator Light Register, a 7 bit MIV Flexo Register, and two Alarm Buzzers. (Ward and Ross 1956) John Ward and I supplied reworked Whirlwind drawings for all these items plus the Whirlwind drawings for the Ferranti PETR and manual tape winder at various times. All of these were forwarded to Stromberg Carlson, San Diego who reworked the designs further and incorporated them with a 19" Charactron for viewing and a 7" Charactron with camera, in the final, delivered installation.

My daily resumes began at about this point, and various phone calls, letters, and visits to MIT by Ed Allmon and various people from Stromberg Carlson are recorded from October 31, 1956 on. Topics included the vector generator and tape winder [R561031]; Ferranti PETR [R56116]; PETR again, plus the fact that they got the contract from AFAC [R561115]; visit regarding Flexowriter, tape reader, and 1103 MIV system [R561126-27]; connectors and pushbuttons [R561210,-11,-12]; calls from their psychologist regarding switch types and layouts (considering a black background for the lights, etc.) [R561218-19]. They finally sent a design study document to us on January 1, 1957, (which I don't have), but still there were more Flexowriter questions a week later [R570118]; the Ferranti automatic turn-off to save motor wear [R57314]; a press release [R57417]; request for a San Diego training program for use of the MIV equipment [R57429]; and finally "the Charactron and MIV equipment turned up at AFAC today" -- on August 14, 1957 [R57814]. But I have no photos or descriptions of what finally was delivered.

A letter from John Ward to John Moser (Strombreg Carlson) on October 24, 1957 congratulates him "for a very fine job" and says we "hope to be making good use of it within a month's time." [C571024]

John Walsh (one of my programmers, and still with SofTech) labored many months on an elaborate 1103 program for extendable, labelled plots of functions for the Charactron as a major part of the AFAC system. (Ross and Ward 1957) But AFAC programmers always were too short of time to do much elaborate MIV programming. The system was indeed used for the Pre-B-58 and B-58 tests, but I don't know any more about it than that. It's claim to fame really is that it was the first working work station explicitly designed for general purpose use.

5.5 MIT MIV for IBM Computers

In the Summer of 1956 preparations were well under way for the installation of the IBM 704 computer in the M.I.T. Computation Center. This also was the time that my Computer Applications Group was officially christened, as we acquired responsibility for the (then unnamed) APT Project in addition to our other projects. My first and second memorandam for the Group (Ross 1956g and i) concerned an assessment of the impact of the 704 on our work. The second memo begins:

"This memorandum is a supplement to Memorandum CA-M-1, 'Servo mechanisms Laboratory Requirements for Computing Facilities 1956-1957'. That memorandum stated that Project DSR 6873, on the development of automatic programming techniques for numerically controlled machine tools, would be transferred to the IBM 704 computer as soon as it became available, but that the work of Project DSR 7138, on the application of digital techniques to airborne weapon systems, would continue to need the facilities of the Whirlwind Computer System because of its dependence on manual intervention techniques. This memorandum considers in more detail the reasons why it is felt that the Whirlwind I Computer must continue to be used, by describing the system of Whirlwind programs which are now being developed for this work, and then considering the modifications to the proposed IBM 704 system which would be required to make this type of system possible using that computer." [p. 1]

Then, after a description of SLURP, it goes on:

"The IBM 704 Computer

"Aside from the large amount of programming which has already been completed, it is apparent that SLURP is independent of the particular computer used, provided it has the appropriate input-output devices. The manual intervention features of the system are central to the philosophy of the system, since without them it becomes merely another elaborate computer programming system. The true significance of the system is that it allows a programmer to conceive of a new method of solution and maintain his momentum and initiative on the problem, unencumbered by the restrictions of the coding system of the computer. In other words, the programmer is permitted to program with programs, and 'program with ideas'.

"It would be extremely desirable to have the larger storage capacity and longer word length of the 704 computer for use in SLURP. It already appears that the capacities of the Whirlwind Computer are being taxed by the size and complexity of SLURP. The present incomplete version contains approximately 10,000 instructions in addition to the 20,000 or so in the comprehensive system, and to this figure must be added several thousand registers required for the storage of a satisfactory amount of data for the problem. Since a large portion of the programming for SLURP consists of generating and interpreting coded information, the short word length of the Whirlwind Computer becomes awkward on occasion, and slows down the operating speed of the system due to increased complexities. Since the Share assembly program and the algebraic coding system being developed for the 704 installation by Computation Center personnel [never carried out] will be more modern and flexible than [p. 4] the Comprehensive System in some respects, and may be expected to expand to include those features of the Comprehensive System which are unique and desirable, it is felt that SLURP efficiency could be greatly improved by the use of those systems. Except for the fact that the IBM 704 will not have manual intervention and oscilloscope-type input-output devices, it would be very desirable, and probably worth the additional programming effort, to transfer SLURP from the Whirlwind Computer to the 704 Computer.

"There appears to be some indication that there is some interest in applying this type of equipment to the 704. The Servomechanisms Laboratory should be active in support of this thinking. Systems such as SLURP seem to be the next logical step in development of improved programming systems for modern computers. The fact that work for Project DSR 6873 for the automatic programming for numerically controlled machine tools is being formulated and solved within the SLURP structure demonstrates that the philosophy of problem formulation and programming which is embodies [sic] in SLURP is by no means restricted to the type of military problems which have fostered its development and for which it is primarily at present intended. The possibility of detailed research into automatic process control and managerial business decisions should provide ample justification for the active consideration of these techniques as an integral part of the MIT Computation Center facility.

"Conclusions

"It is felt that the problems presently being considered in the Servomechanisms Laboratory cannot be successfully solved without the use of SLURP so that until an equal facility becomes available elsewhere this work is committed to the Whirlwind Computer. It is felt also, however, that the concept of systems such as SLURP is a significant advance in the use of computers as data processing devices, and that active consideration should be given to this type of system as an addition to the facilities of the MIT Computation Center in the near future. It would be necessary to expand the equipment planned for that facility to include manual intervention input devices and oscilloscope-type output devices in order to realize this type of system. The experience gained from the use of this type of equipment in this application in the Servo Lab SLURP system, as well as the experience of the Sage system development, should provide an adequate foundation for the planning of a Computation Center facility of this type. The potentialities of this type of a system as a research tool for experimentation in the newly unfolding area of

general data processing are very great. It is felt that it would be in the interests of the Institute as a whole to pursue this line of thought." [p. 5]

My resumes show that on November 1, 1956, F.J. Corbato (still a professor at MIT) "called to inquire about Servo Lab people giving talks in Digital Computer Center [sic] seminars. He is now on their staff and is in charge of these seminars. I said that we might talk about SLURP and our general problem solving system." [R56111] On November 6 "should possibly consider obtaining support from Mission Director Project [a new component of John Ward's Bomber Defense project] for attaching MIV equipment to the 704 ... Will IBM let such equipment be attached to the computer and also how do we get enough computer time to make the investment useable?" [R56116] This was a problem we later solved by joining as a co-sponsor of the MIT Cooperative Computer Lab's IBM 709 in Building 10 in the early 1960s.

On November 27 I gave my Comp Center talk on "SLURP: An Experimental Human-Computer System for Problem Analysis and Solution" [R561114], and the interesting note is that Wes Clark (then doing TX-0 at Lincoln) suggested, in the discussion, "that the switches be programmed on the tube and thrown by means of a light pencil [sic]. In this way a programmer can do all of his own human engineering of switch layouts as well. [We later called this "light buttons" (Ross and Ward 1961, p. 80).] Slurp is already set up so it could do this operation merely by writing an appropriate Slurp program. Wes Clark would like to have several copies of the Gestalt paper and I will plan to take them out when we visit TX-0 tomorrow." Also at the talk, Frank Verzuh (Director of the MIT Comp Center) "said he was interested in talking with me about getting this kind of facility for the Computation Center" [R561126-27]. And the next day "Gave a stack of Gestalt papers to Wes Clark. He seems to be quite interested in what we are doing and there may be a possibility of pooling our interests if not our efforts in making TX-2 specifically designed for Slurp type of problem solving. Should plan to discuss this with him and his group in the near future." [R561128] We did collaborate on light pen design and improvement, under John Ward, after TX-0 was moved to MIT (I heard it was scheduled to move on June 27, 1957 [R57627]), but actual collaboration never took place.

Not much happened during 1957 on these ideas. TX-0 moved from Lincoln to ESL, with Earl Pugh in charge. In September, Whirlwind "passed into Lincoln's hands" [R5794-9] and we continued our developments there and on the AFAC 1103, while doing APT work also on the 704. By year end "A summary-progress report for the mission director is scheduled ... [Writing about SLURP will serve to] explain the whole problem-solving philosophy which I have been evolving over the last several years. [This will serve] as powerful ammunition for battling to keep Whirlwind going for a long time." [R57124-58019]. I cover the Gestalt Programming to SLURP transition in the next section, briefly.

In January 1958, Peter Elias (then EE Department Head) suggested to John Ward that I should get together with John McCarthy, Marvin Minsky, and Richard Marcus (all then at MIT's Research Lab for Electronics, RLE) about my use of MIV Techniques, [R58019-28] and John Ward wrote me a memo to that effect. (Ward 1958) John's memo says Dick "is working actively on function display under Manual Intervention Control." Later Dick came to work in my group and still is at MIT. By January 30, I finally arranged to have a lunch with McCarthy, which then transferred to Minsky's home in mid-afternoon, "and we talked some more until around 2:30 in the morning. Oliver Selfridge's group at Lincoln with whom Minsky works has $25,000 a month which must be spent on equipment, not manpower ... They are presently thinking of getting the [standard IBM] real time package for the Lincoln 704 and are interested in investigating whether the Convair MIV package that we worked out for AFAC would be within their budget." [R580131] I don't know whether Dliver contacted Convair directly or what he ended up doing, but this session did begin a long, sporadic interaction on equipment matters with John and Marvin. A week after our long session I loaned John McCarthy "our information on the Stromberg Carlson [Convair] MIV equipment ... informed him of our present investigations and said .. perhaps he would be a good one to work on the automatic programming features since he has been working quite a lot on a super compiler with all sorts of logical statements, possible in it which may fit in well with the system we have been working on. We shall see." [R5827] LISP was in the wind, and I was in the midst of APT, of course.

Again nothing happened regarding MIV equipment for months, although a talk I gave (on APT) at IBM Kingston Labs evoked brief interest on their part in scope input and MIV for their IBM 704. [R58815-20] In September, Dick Bennett (who then had a consultant company) was under contract to Lincoln to upgrade the Whirlwind Utility System, and he thought Frank Heart's group "also should be interested in being able to operate everything from the Room 222 console ... We also discussed ... putting things under direct typewriter control." [R58924] But most of Dick's plans were not carried out.

Even though the preparation for the APT press conference were pressing, I was concerned about the facilities for follow-on work, so in early January 1959 I said

"Whirlwind definitely is going to close on June 30 and since it is poor politics to fight it at this time, we are going to go along with it, so that we have to find a substitute. My thought is that by combining TX-0 and 704 we could have a really fancy facility since TX-0 with its newly expanded memory, which is not in yet, of 8,000 words should be able to do [display and light pen support.] ... We are having a meeting .. between [Dean] Arden, Jack Gilmore, John McCarthy, myself, Arnie Siegel, and [?] who will be in charge of the Lincoln 709

programming to consider using the real time package [IBM's terminology] on the 709 to simultaneously service about 20 Flexowriters in TX-0 fashion. ... We are going to start off with one Flexowriter on the 704 real time package." [R581229-59019] This was the start of MIT's time sharing developments.

In January 1959, "Attended a meeting in Dean Arden's office concerning 704 Flexowriter input ... I didn't feel that the group accomplished very much." [R590112-21]

In early March, "John McCarthy is calling a series of meetings to consider operator and compiler programs for the 7090 computer which will be the transistorized version of the 709 which will replace the 704. We had a meeting yesterday [March 3]." I spoke about MIV and Group Control. "I am hoping that in later sessions [we can show that MIV] should be an integral part of the programmer's bag of tricks applicable to any problem." [R59219-34] But by the end of the month "The meetings to discuss 7090 problems seems to have lost momentum, so it is unlikely we will have another one very soon." [R59323-331]

In April "With the imminent breakup of Whirlwind, we have started to consider putting a good manual intervention console on the 704. I have now decided to soft pedal the connection of TX-0 to 704 until we have a definite use. We got together with John McCarthy and Herb Teager who is now putting the Flexowriter on the 704. [I believe Arnie Siegel, who knew Whirlwind well, had done some early preparations to adapt our earlier work.] John Ward and Don Clements are going to assist in this and it looks like our first move will be to get a light pen hooked up to the 704 to use the present scope ... In the long term, however, we are thinking of developing a self maintaining display system, [Later the ESL Display Console (Ross and Ward 1961, pp. 79-92), which was Rob Stotz's Masters thesis, came from this idea.] possibly built out of Whirlwind components. There is quite a bit of engineering involved there, however. McCarthy and Teager are going over to see the Whirlwind equipment with us at 11 a.m." [R59416-511].

On April 16-18, 1959, a select conference on Symbolic Manipulation (organized by McCarthy, Perlis, and Newell [R59219-34]) was held at MIT. (McCarthy 1959) This was the first public presentation of LISP by McCarthy, COMIT by Yngve, and "I talked for a short time on multi-mode control as applied to list searches, group control, and proposed a modified list structure which seems more appropriate to our design machine application." [R59416-511] This was my "reversed index registers" (as I later called it) method of general pointers and "n-component elements" of Plex Programming, (Ross 1961) which began what now is the field of abstract data types. Evidently Al Newell made no presentation at MIT, but "On May 4, 5 and 6 I attended a conference on self organizing systems ... in Chicago. [Some papers] were quite good, ... [especially] one by Allen Newell on the GPS system general problem solver that he is working on with Simon and Shaw. ... will plan to get together with him." [R59416-511].

On May 12, I took McCarthy, Teager, and Don Clements (then project engineer for the APT Project) "to look over the Whirlwind equipment. They seemed satisfied enough but our present thoughts are to find out what is going on at Lincoln, too." [R59512 & 13] A memo from me (Ross 1959) shows that Arden, Clements, McCarthy, Minsky, Teager, Ward, and Ross visited Lincoln (re 709, Charm, TX-2, and SAGE) on May 19, but "Our meeting out at Lincoln ... was pretty much of a farce. People were late or missing or not very sympathetic ... so not much was accomplished." [R59518-20]

5.6 The Design Machine

So as 1960 rolled around, various seminal equipment developments at MIT had been initiated, but they are better treated as components of the history of time sharing, computer graphics (hardware and software), computer-aided design (CAD), software engineering, and software technology, rather than merely as extensions of the personal work station theme of this paper. I hope to be invited sometime in the future (though not too soon, for these papers are excruciating torture to write!) to contribute to the chronicling of those development, as well.

Any such future consideration of CAD work stations would have to begin with a mention of George Price's prescient article in the November 1956 issue of Fortune Magazine, (Price 1956) and since it, too, was a "first in the fifties", I must reference it here.

Going back again in time, in January 1957 I said

"Bill Webster [Air Force contract monitor for APT] also had a copy of the article on a design machine which I had been told about in New York. It's amazing the similarity between what is in that article and what was in my Gestalt paper and what we have been working on the past year or so. He even has pushbutton language with sentences of the form that we are planning for our milling machine, and the computer drawing pictures of the part being designed on the scope just like our scope plot, etc. I plan to write him a letter describing our work and enclose a copy of my paper and send a copy of this correspondence to the editor of Fortune and see what results. I think it is quite interesting that we have been actually carrying out what in his article is merely a proposal which everybody feels very strongly isn't as wild as it seems. Webster wanted to check with me that his impression that we were doing just what the article called for was correct, and I assured him that it was." [R57017-8]

I finally got a library copy to read by January 25 [R570125], and on February 6 wrote to Price, enclosing a Gestalt paper reprint, and

describing our APT progress, then being issued as the first APT Interim Report. I then go on:

"These routines provide the necessary mathematical structure around which a convenient human language can be built to make a complete automatic programming system. The detailed specification of this human language has not been attacked yet, but it will be the major focus point for the efforts of the group. The language will have specially designed features for description of surfaces and their interrelation, as well as for the instruction of the machine tool itself. The form of the language will probably be similar to that which you propose for the design machine, and as outlined in my paper, although it will be a written language and not use push buttons initially [-- used in the Gestalt paper and in Price's article] ...

"Another important part of the language problem is the computer-to-human language, which we envision in two forms. First there will be the ability of the computer to 'talk back' to the human in the same type of language which the human uses, probably in a written form. The computer's language also will be of a pictorial form in which the computer will draw pictures for the human to check and work from in a way almost identical with your proposals in the FORTUNE article. A rudimentary routine of this sort has been in operation on the Whirlwind I Computer for several months now, which draws on the output oscilloscope of that computer arbitrarily positioned and scaled axonometric projections of the part being made. True perspective will be programmed later if it seems warranted. At present the picture consists of the sequence of 'cut vectors' which are used to program the machine tool, but more elaborate schemes are planned. At the present time the pictorial displays are photographed and no effort has been made to increase the speed of the program so that a cycled display can be viewed easily on the display tube itself. Memotron-type tubes are not presently installed on the Whirlwind Computer although they would be very desirable for this and other applications. [The ARDS Display of Rob Stotz was the first such memory tube display (Ross and Ward 1961, pp. 100-113).]

"The striking similarity between our work and the proposals of your FORTUNE article indicate that we have a number of very strong mutual interests and a similar approach to these problems. Mr. William Webster of the Air Materiel Command, who is our project monitor, brought your article to my attention and showed me a letter which you had written to him. In that letter you mentioned a study which you made concerning a design machine using the IBM 704 computer as a major component. I would very much appreciate a copy of this report if you have any available in order that we may know in more detail your ideas on this subject. We plan to transfer our work to the IBM 704 in the very near future.

"I look forward with anticipation to further correspondence with you on this most interesting subject." [C5726]

A February 25 reply from Price enclosed a reprint of the FORTUNE article and his retyped "supplementary memo", but said he was changing jobs and moving, so he hadn't "been able to find the time to do more than take a quick glance a your two papers." [C57225] On May 13 he visited my home and we "talked for an hour or so ... but we never did get around to talking the same language or really discussing anything ... He has no technical training in this field." [R57513] But he certainly wrote an interesting article, and his Design Machine proposal surely was a first, including many features that later were used in actual CAD work stations. As a side box [p. 153] about "The Author and His Machine" said:

"Price began thinking about his Design Machine some ten years ago. Could the Machine be built today? One leading computer expert shown the proposal was skeptical. Price thereupon prepared a detailed supplementary memorandum demonstrating how an IBM 704 computer could be incorporated into a Design Machine, how a complex part could be described to the Machine, and how with the aid of certain auxiliary devices the Machine would display the part -- in 3-D. Computer experts at IBM pronounced the memorandum eminently sound. The first skeptic conceded that Price had shown how the job might be done."

The main thrust of the article was "eight recipes for modernizing R. and D.," one of which was the Design Machine, and another to mechanize "reading" of technical literature for mechanized filing (with a reference to V. Bush and the Patent Office). Besides describing the pushbutton and joy stick preparation of stereoscopic pictures of parts and "invisible models -- recorded in a magnetic memory," with "preparation of control tapes for guiding automatized machine tools," the article says "a single Machine might be shared among several companies scattered around the country, for a number of control stations can be in use simultaneously, with keyboard signals being briefly stored on magnetic tape, and the central computer switching from one station to another every few seconds." [p. 228] [-- an idea also promoted by Jay Forrester for Whirlwind, in 1948 (Redmond and Smith 1980, pp. 233, 234).] Price's article was well ahead of its time.

Section 6

SYSTEM SOFTWARE

As I said at the beginning, a final essential ingredient for a viable, general, PWS scheme is a systematized way to link the work station capabilities to the class of problems to be solved through some form of software framework. As I have indicated, the earliest such framework, when Whirlwind had only 2K words of memory, took the form of several programming principles, such as "store switch values in variables", the language design principles of the Gestalt Programming paper, and the rudiments

of drum/ES storage management prompted by the Mistake Diagnosis Routine (MDR). But with the advent of core memory, and its expansion to 6K words, and with the growing sophistication of approach engendered by early efforts at MIV in the Evaluation Program, a truly systematized solution was the natural evolutionary step.

6.1 The Servo Lab Utility Routine Program

The first coherent expression of these ideas appeared in (Ross 1956h) [note -- when APT work was just getting under way, as well], which said:

"After some experience with the integration of the manual intervention facilities into the previously fully automatic evaluation programs, by the Servomechanisms Laboratory, it was realized that the manual intervention features could be used as a direct aid to programming by incorporating a number of general-purpose utility routines into the system. Shortly thereafter, the new task concerning the investigation of the techniques and problems for an airborne mission director was undertaken. Since detailed specifications on the mission director problem were not then available, and also since the committment of the evaluation studies to the B-58 program made progress in that direction imperative, it was decided that a worthwhile merging of these two interests could be achieved. This was done by commencing the elaboration of the evaluation program on the Whirlwind computer into a general purpose problem-solving system combining the talents of the human programmer and the electronic computer for the solution of general computer problems. The result is a human-computer system which may be viewed either as an elaborate evaluation program, or as a prototype problem-solving program with the problem of evaluation of airborne fire-control systems as the motivating core. This memorandum stresses the latter viewpoint. [p. 1] ...

"The combined evaluation and problem-solving system which is the subject of this memorandum has been given the name, the Servo Lab Manual Intervention System (MIV System). The main feature which distinguishes this system from other computer programming systems is that all of the major facilities of the system are instantaneously available under manual intervention control. [p. 3] ...

"The facilities of the MIV System may be grouped into three categories according to their primary function and motivation. Some of the features are concerned with the problem itself (in the present case the evaluation program). Other features are strictly of a utility nature having to do with input and output of data in various tabulated and graphical forms. The third category concerns features which arise almost entirely from the fact that a manual intervention system is being used and have primarily to do with reliability and ease of operation of the overall system. The main features of each of these categories will be described in turn." [p. 3]

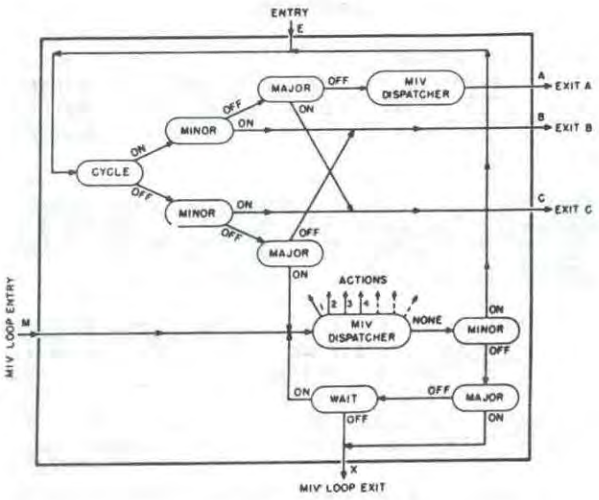
Five more pages then detail the plans for what came to be called "SLURP", just nine days later, in the second Computer Applications Group memo, mentioned earlier. (Ross 1956i) The one-page summary of features, some of which were implemented only in later years, was:

"The Servo Lab utility routine program (SLURP) is a system of Whirlwind Computer programs which combine the manual intervention features of that computer, with modern programming techniques, into a unique problem-solving system. SLURP includes all of the facilities of the Comprehensive System developed for the Whirlwind Computer by the Digital Computer Laboratory staff, plus a large number of general purpose routines for extracting and analyzing information about arbitrary computer programs. The governing philosophy of the system is to allow the programmer to work on one small part of a very large and complicated problem, virtually independent of the computer which is being used and those aspects of the overall problem which are not of immediate concern. The major features of the system which make this possible are as follows:

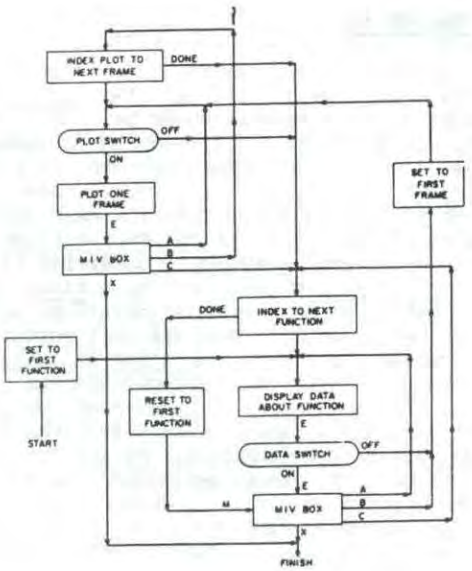
"1. A group control program which allows the automatic incorporation of new sections of programming into the system.
"2. A manual intervention (MIV) system of programs which allow the programmer to interrogate and instruct the computer with respect to the overall problem in terms of a specially designed and easy-to-use language represented by: switches, buttons, lights, and visual displays.
"3. A logging program which records all of the MIV actions taken by the programmer.
"4. An editing program which edits the logged information into an easy-to-read record of the manual actions taken, and generates a log playback tape.
"5. A log playback program which automatically simulates manual actions in response to the instructions on the log playback tape.
"6. An elaborate set of plotting and tabulating programs for data presentation and record keeping.
"7. A mistake diagnosis routine (MDR) which may be used to abstract arbitrary intermediate computed quantities from any program and present these quantities in any of the above forms [including scope].
"8. The SLURP program proper which is a simulated, generalized, special-purpose computer which allows the incorporation of all of these routines into a smoothly functioning system, along with the capacity for continued expansion of these facilities." [p. 3]

6.2 The SLURP Simulated Computer

The SLURP computer had many features that independently were discovered and incorporated into the Burroughs B5000 computer (I was astounded to see the similarities when I first



Definition of MIV Box re-entrant subroutine of Servo Lab Utility Routine Program (SLURP) simulated computer — basis for systematized MIV languages.



Hierarchic control of complex program action with generic MIV Box controls. Precussor to current mouse/menu control scheme.

read their manuals in the early 1960s). SLURP included built-in Group Control (memory paging), the Memory Table (virtual addresses for re-entrant program segments), Alarm Conditions (exception conditions with optional MIV and MDR facilities), and the ZIP Interpretive Program (which "serves the important function of establishing the MIV language which is appropriate for a particular program" -- and integrated all of the other features, including I/O in a powerful, very condensed macro-command-like language).

A unique feature of SLURP was its "multi-mode control element". (Ross 1958) As the definitive report on SLURP says:

"The instruction repertoire of the SLURP computer contains no mathematical instructions but instead concentrates on a wide variety of a) jump instructions for transferring control from one sequence of instructions to another, and b) instructions which control selection and adjustment of input-output equipment. Individual flexowriter typewriter characters are included in the instruction repertoire and the execution of this type of instruction involves the reproduction of these characters on a selected output device. A SLURP instruction may occupy an arbitrary number of Whirlwind registers. There are two binary digits called the A and C bits which are set aside in each instruction and are used to label individual instructions for execution in the various modes of the control element ...

"If the A bit is ONE and the C bit ZERO, the instruction will be executed when the control element is set to the A mode but will not be executed if the control element is set to the C mode [etc.]. All SLURP instructions (with the exception of a few jump instructions) are executed when the control element is set to the B mode, independent of the settings of the A and C bits. Since the mode of the control element may be set by jump instructions, the setting of the control element may be changed as often as necessary to accomplish a desired result ...

"In addition to the return jump instruction for termination of a remember jump sequence, it is also possible to remember jump to a flexo phrase (any sequence of characters or words), and the occurrence of a stop character in that flexo phrase will act as a return jump under certain circumstances. However, if that same flexo phrase is executed by encountering it in the normal sequence of operations (instead of by an entry via a remember jump), the stop character is ignored. In this way a lengthy flexo phrase which contains useful subphrases or words may be extracted from the entire sequence by means of remember jump instructions." [pp. 3-82 to 3-84]

This feature was used to have abbreviated labels for display and long labels for printouts, and the same technique gave complex behavior if the "words" were other instructions. Our favorite test case

THE SKUNK SAT ON THE STUMP
THE SKUNK THUNK THE STUMP STUNK
THE STUMP THUNK THE SKUNK STUNK

took only 13 SLURP instructions including output device selection. [p. 3-85] SLURP packed a tremendous punch in very little space.

6.3 The MIV Box

One of the most powerful features of MIV language design with SLURP was the MIV Box, a re-entrant subroutine that could be used in many places. [p. 3-127 to 3-137] The basic language idea was that "if the human does not like what is presently going on, he merely needs to remember to depress an appropriate exit button. The appropriateness of the two exit buttons can be given a universal meaning by referring to one as a major exit, and the other as a minor exit. Then no matter what program is operating, if the human is unsure which of the two exit buttons to press, he may first try the minor exit and if that does not give the desired action, then pressing the major exit is guaranteed to work. ... It is not necessary to know beforehand all the possible actions which may be controlled by exit buttons if it can be established that those programs will operate on a hierarchy of actions. The major and minor exits can then be associated merely with a change in level within whatever hierarchy exists for a particular program. This uniformity of meaning of language is achieved by combining the MIV Dispatcher, the wait switch, the major and minor exit buttons, and one new switch, the 'cycle' switch." The example shows how only two more switches allowed many choices of control for selecting and plotting successive frames of a sequence of functions, as in the Evaluation Program. One more switch could similarly specify console display only or off-line camera, as well, and the SLURP system would efficiently see to the details.

CONLCUSION

I hope that I have succeeded in defending my thesis that there were indeed some significant PWS firsts in the Fifties, even though only a tiny fraction of the computer resources that today are thought to be essential were then available. The revolutionary evolution since those early days has indeed brought much greater sophistication and broader capabilities -- including the all-important hardware, software, and systematized approach improvements which allow ubiquitous spread to all types of users. But don't sell the old days short. Big ideas can come in small packages. Maybe that's an idea we've lost track of in today's technology where we shrink things mechanically to make them smaller. Maybe we need a sharper return to those earlier days when the only recourse was to think big things into their distilled essence -- and make them work.

THANKS

... to the MITRE Corporation Archives and the MIT Museum for the Whirlwind photos.

... to MIT and my many colleagues for making it all possible.

REFERENCES

AMERICAN ORDNANCE ASSOCIATION (1955) April 14-15. Symposium on Proving Ground Instrumentation Problems, Air Force Missile Test Center, Patrick Air Force Base, Cocoa, Florida (program and papers).

BRACKETT, C.F. (1959) February 2. The Direct Flexowriter Keyboard Input System at Whirlwind I (Barta). Lexington, MA: Division 2 - Lincoln Lab. Interoffice Memo (no number), 3 pp.

CHARACTRON PROJECT (1955). Charactron Display Console Model 708 (January 24, 1955, 14 pp.); Charactron Computer Readout System, Model 100 (June 3, 1954, 22 pp.); The Charactron Rapromatic (no date, 11 pp.). San Diego, CA: Convair division of General Dynamics Corp.

COHEN, ARNOLD (1985) October 4. Oral comments to D.T. Ross at Annual Meeting of the Charles Babbage Foundation.

CONVAIR (1954) August 9. Agenda for Flight Test Program Meeting see also Ross (1954).

EJCC (1955) November 7-9. Program of the Eastern Joint Computer Conference and Exhibition, Statler Hotel, Boston, 27 pp.

ELECTRONIC DESIGN MAGAZINE (1956) May 1. New York, NY: Hayden Publishing Co. Computer Developments: Design Trends from Meetings, pp. 36 & 37.

ERA (1954a). ERA 1103 General Purpose Computer System (brochure). New York, NY: Engineering Research Associates Division of Remington Rand, Inc., 4 pp.

ERA (1954b) January 5. ERA 1103 Installation Requirements, St. Paul, MN: Engineering Research Associates Division of Remington Rand, Inc., 16 pp.

FORRESTER, J.W. (1948) July 29. Whirlwind High Speed Computing. Cambridge, MA: MIT Servo Lab. Rep. No. R-142. Appendix A in Redmond and Smith (1980), pp. 225-236.

HAMILTON, D.A. (1955) August 4. Conventions for Presentation of Results from Whirlwind I Polynomial Fit Program. Cambridge, MA: MIT Servo Lab. Memo 7138-M-116, 4 pp. See also Ross (1954), and Ross Thompson, and Cundiff (1956).

HOLMES, L.L. (1956) April 3. Flexowriter Input to WWI. Lexington, MA: Division 6 - Lincoln Lab. Interoffice memo, 2 pp.

McCARTHY, J. (1959) February 27. Invitation to Conference on Symbol Manipulation Systems, April 16-18, MIT.

MIT DIGITAL COMPUTER LAB. (1956) March 1 and 2. Seminar on Micrprogramming (Announcement and agenda). Cambridge, MA. 2 pp. plus map. June 6, 1956: Partial transcript of D.T. Ross presentation pp. 152-161, via F.W. Helwig.

MIT SERVO LAB. (1952-1958). Certain Lists of laboratory personnel.

MIT SERVO LAB. (1955) March 8, 9. Symposium on Design and Evaluation of Bomber Fire-Control Systems.

MIT SERVO LAB. (1958) December. Data Reduction Programming for Pre-B-58 Tests of the XMD-7 Fire-Control System (3 Volumes). Cambridge, MA: MIT Servo Lab. Reports: Ross, D.T. Functional Description of the Data Reduction System (7886-R-1), 44 pp; Scheff, B.H. Initial Data Processing (7886-R-2), 149 pp.; Ross, D.T. and McAvinn, D.F. Evaluation of Fire-Control System Accuracy, (7886-R-3), 177 pp.

NIELSEN, K.L. and HEYDA, J.F. (1951) July 20. Mathematical Theory of Airborne Fire Control. Washington, D.C.: U.S. Government Printing Office, NAVORD Report 1493, 208 pp.

PRICE, G.R. (1956) November. How to Speed Up Invention. New York, NY: Fortune magazine, p. 150 ff.

PRICE, G.R. (1957) February. Technical Details Concerning the Design Machine. Kingston, NY: personal memo, 72 pp.

PROJECT WHIRLWIND (1952-1957) Quarterly Summary Reports No. 31 - No. 50. Cambridge, MA: MIT Digital Computer Lab.

REDMOND, K.C. and SMITH, T.M. (1980). Project Whirlwind: The History of a Pioneer Computer. Bedford, MA, Digital Press, 280 pp. Index.

ROSS, D.T. (1952) July. Autocorrelation Program working papers.

ROSS, D.T. (1953a). First Evaluation Program Flow Diagram. Computation Notebooke pp. 26 & 27.

ROSS, D.T. (1953b) June 27. An Introduction to the Use of Air-Mass Ballistic Tables in Airborne Fire-Control System Evaluation (CONFIDENTIAL, Title Unclassified -- declassified November 1958). Cambridge, MA: MIT Servo Lab. Rep. No. 6506-ER-46, 25 pp.

ROSS, D.T. (1953c) November 24. A Mistake Diagnosis Routine for Whirlwind I Programs. Cambridge, MA: MIT Servo Lab. Rep. No. 7138-R-1, 26 pp.

ROSS, D.T. (1954a) May 24. Missile Launch Simulator program for Memory Test Computer working papers.

ROSS, D.T. (1954b) June 25. Improved Computational Techniques for Fourier Transformation. (MS Thesis). Cambridge, MA: MIT Servo Lab. Rep. No. 7138-R-5, 84 pp.

ROSS, D.T. (1954c) July 16. Letter to W.J. Moe of Engineering Research Associates (ERA) regarding 1953 Polynomial Fit Program, 2 pp. See also Hamilton (1955).

ROSS, D.T. (1954d) August. Scope Input Program (no original; revised copy fc Tape 126-50-505, 2/15/55).

ROSS, D.T. (1955a) March 8. Handwritten instructions to DAH for RDSS DEMO. Director Tape, plus printout of same.

ROSS, D.T. (1955b) June. A Basic Input Translation Program for the ERA 1103 (program, runs, and working papers).

ROSS, D.T. (1955c) July 25. Special In-Out Equipment for Human Participation in High-Speed Computer Operation. Cambridge, MA: MIT Servo Lab. Memo 7138-M-112, 8 pp.

ROSS, D.T. (1955d) October 13. [The] Gestalt [System of] Programming -- A New Concept in Automatic Programming, handwritten draft, 15 pp.

ROSS, D.T. (1955e) November-December. Handwritten draft pages.

ROSS, D.T. (1955f) November 17. Gestalt Programming: A New Concept in Automatic Programming (second draft, typed) 12 pp. plus addendum: "Proposed Changes for Final Draft", 2 pp.

ROSS, D.T. (1955g) December 14. Proposed Equipment Modifications for the Pre-B-58 Data Reduction Programs. Cambridge, MA: MIT Servo Lab. memo 7138-M-128, 7 pp.

ROSS, D.T. (1956a) January. Handwritten oral presentation drafts and notes.

ROSS, D.T. (1956b) February 7. Gestalt Programming: A New Concept in Automatic Programming. New York: Proceedings of the Western Joint Computer Conference. AIEE for the JCC (now AFIPS), pp. 5-10. Also Cambridge, MA: MIT Servo Lab. Rep. No. 7138-TM-7, 14 pp.

ROSS, D.T. (1956c) March 1. Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-144, 6 pp.

ROSS, D.T. (1956d) March 14. Flexowriter Keyboard Input to WWI, Preliminary Specifications. Cambridge, MA: MIT Servo Lab. Memo 7138-M-148, 3 pp.

ROSS, D.T. (1956e) April 5 - Summer 1956. Proposed Microprogramming Instruction for Whirlwind I Computer. Working papers, drawings, and memo outline.

ROSS, D.T. (1956f) April 30. A Proposed Flexowriter Input for the Whirlwind I Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-160, 4 pp.

ROSS, D.T. (1956g) September 26. Servomechanisms Laboratory Requirements for Computing Facilities, 1956-1957. Cambridge, MA: MIT Servo Lab. memo CA-M-1, 4 pp.

ROSS, D.T. (1956h) November 14. Programming Progress for Mission Director Task. Cambridge, MA: MIT Servo Lab. memo 7138-M-200, 8 pp.

ROSS, D.T. (1956i) November 23. Whirlwind Versus 704 for Servo Lab Problems. Cambridge, MA: MIT Servo Lab. memo CA-M-2, 5 pp.

ROSS, D.T. (1958) April. A Multi-Mode Control Element; A Philosophy of Problem Solving; The SLURP System for Experimental Programming. Cambridge, MA: Research in Defense Techniques for Airborne Weapons: 1957 Annual Report, Volume 2. MIT Servo Lab. Rep. No. 7668-R-5(2), pp. 3-81 through 3-163.

ROSS, D.T. (1959) May 13. Visit to Lincoln Lab., Tuesday, May 19. Cambridge, MA: MIT Servo Lab. memo (no number).

ROSS, D.T. (1961) March. A Generalized Technique for Symbol Manipulation and Numerical Calculation. Communications of the ACM 4(3): 147-150.

ROSS, D.T. (1977). Origins of the APT Language for Automatically Programmed Tools. New York: History of Programing Languages (R.L. Wexelblat, Ed.), (1981) Academic Press, pp. 279-367.

ROSS, D.T., Thompson, D.A. (Hamilton), and Cundiff, T. (1956) December 17. Polynomial Fit Program. Cambridge, MA: MIT Servo Lab. Memo 7138-M-202, 18 pp. See also Hamilton (1955).

ROSS, D.T. and Ward, J.E. (1956) May. Investigations in Computer-Aided Design for Numerically Controlled Production (Final Report for 1 December 1959 - 3 May 1967). Cambridge, MA: MIT Electronic Systems Lab. for Air Force Materials Lab., WPAFB, Ohio. Rep. No. AFML-TR-68-206. MIT Rep. No. ESL-FR-351, 229 pp.

ROSS, D.T. and Ward, J.E. (1957) October 22. Charactron Plot Program for Pre-B-58 Data Reduction. Cambridge, MA: MIT Servo Lab. memo 7668-M-251, 4 pp.

ROSS, D.T. and Ward, J.E. (1961) January. Picture and Pushbutton Languages (Chapter VIII), and Ward, J.E. A Manual Intervention Facility (Chapter IX), in Investigations in Computer-Aided Design, Interim Report for December, 1959 to May 30, 1960. Cambridge, MA: MIT Electronic Systems Lab. Rep. No. 8436-IR-1, 156 pp.

WARD, J.E. (1954) July 23. Disposal of Mechanical Correlation Computer (MCC). Cambridge, MA: MIT Servo Lab. Memo 7138-M-49, 2 pp.

WARD, J.E. (1955) May 2. Symposium on Data Reduction (Announcement). MIT Servolab (no number).

WARD, J.E. (1958) January 27. Manual Intervention. (Memo to D.T. Ross.) Cambridge, MA: MIT Servo Lab. (no number).

WARD, J.E. (1960) January 15. Automatic Programming of Numerically Controlled Machine Tools (Final Report). Cambridge, MA: MIT Electronic Systems Lab. Rep. No. 6873-FR-3, for June 26, 1956 to November 30, 1959. Reprints 6873-FR-1 (July 30, 1952) for July 1949 to July 1954, [sic] and reprints 6873-FR-2 (March 15, 1956) for July 1949 to March 15, 1956.

WARD, J.E. (1970) June. Computer-Aided Design for Numerically Controlled Production (Final Report for 1 May 1967 - 30 January 1970). Cambridge, MA: MIT Electronic Systems Lab. for Air Force Materials Lab., WPAFB, Ohio. Rep. No. AFML-TR-70-78. MIT Rep. No. ESL-FR-420, 121 pp.

WARD, J.E. and Ross, D.T. (1955) July 29. Minutes of M.I.T. Data Reduction Symposium: June 1 and 2, 1955. Cambridge, MA: MIT Servo Lab Rep. No. 7138-S-1, 20 pp.

WARD, J.E. and Ross, D.T. (1956) March 1. Suggestions for Manual Intervention Facilities on the Air Force Armament Center ERA 1103 Computer. Cambridge, MA: MIT Servo Lab. Memo 7138-M-144A, 30+ pp.

WARD, J.E. and Ross, D.T. (1956) May 8. Draft paper of Revision of (Ross 1956c), incorporating MIT Digital Computer Lab. memo and drawings. Cambridge, MA: MIT Servo Lab. memo 7138-M-144A.

WJCC (1956) February 7-9. Program of the Western Joint Computer Conference and Exhibit. Fairmont Hotel, San Francisco, 23 pp.

WOLF, W.M. (1954) September-November. Worksheets and photos for Large Letter program. Flow diagram and WWI code for Manual Intervention tests.

Some Reflections on Early History

J. C. R. Licklider
Massachusetts Institute of Technology

Abstract

The ideas and developments reviewed briefly are mainly from six sources: (1) the M.I.T. Lincoln Laboratory during the middle and late 1950s, (2) Bolt Beranek and Newman in 1957-1962, (3) the ARPA program in 1962-1964, (4) IBM Research in 1964-1966, (5) Ivan Sutherland's lab at Harvard in 1967, and (6) Project MAC at M.I.T. in 1968-1972.

For instance, there were (1) cloud filter stations with light guns and light pens, (2) a single 1024x1024 scope display time-shared four ways and a workstation for studying technical documents, (3) Rand tablets, Teagher tables, and Bernstein's horizontal text-editing display onto which both the human editor and the computer editor could write (and understand each other), (4) a computer system especially for computer-based education and a powerful vector graphics display system, (5) a PDP-1-based, single-user workstation with four display screens, a paper tape unit, a magnetic tape unit, a printer, and the beginnings of Sutherland's virtual space displays, and (6) graphics over the ARPANET and a multi-window workstation based on the Evans and Sutherland LDS-1.

One of the key insights to be offered explains why workstations have vertical instead of horizontal (desk-like) display surfaces. Part of the plot is that, when, in the 50s or 60s, a cathode-ray tube was mounted face-up, some of the phosphor fell down onto the electron gun and caused a short circuit. Another part is that Fredkin and DEC simply got mixed up about which way the PDP-1 display ought to swivvel.

limitation Issues - what discuss?
Instrumentation
Input devices
output devices { quality, size
Software for users; learnability
Workstation DC Xerox
Documentation as part of pws
SW with group capability; convenience of use

system for studying stuff

The ARPANET & Computer Networks

Lawrence G. Roberts
NetExpress Inc.

In 1964 only large mainframe computers existed, each with its own separate set of users. If you were lucky the computer was timeshared, but even then you could not go far away since the terminals were hardwired to it or connected by local phone line. Moreover, if you wanted data from another computer you moved it by tape and you could forget wanting software from a different type of computer. Thus, most users were tied by their computer and terminal to a very restricted enviornment.

Today, in 1985, your terminal could well be a microcomputer networked with a very large, worldwide collection of other computers. You can obtain data and software from all these computers relatively easily with room for improvement) or, where convenient, use the software and data on its home computer by remote access, computer to computer.

This change, which has occurred over the past 20 years, is in part a massive and evolutionary change in computer technology, and in part a modest and revolutionary change in communications technology. The revolution in communications started with an experiment in computer networking, the ARPANET, and grew into a communications revolution called packet switching. Today virtually all the world is linked by packet switched communications service so that any terminal can access almost any computer in the world. This packet switched data network has grown up independent of the telephone network, but over the next 20 years the basic fabric supporting all switched services (data, telephone and video) appear likely to become converted to packet switching, completing the revolution.

History of Network Concepts

Going back to examine the history of computer networks, the first event for me took place in November 1964 at the Second Congress on the Information System Sciences in Hot Springs, Virginia. In informal discussions with J. C. R. Licklider, F. Corbato, and A. Perlis, I concluded that the most important problem in the computer field before us at that time was computer networking; the ability to access one computer from another easily and economically to permit resource sharing. That was a topic in which Licklider was very interested and his enthusiasm infected me. My interest was more toward the networking and communications issues rather than the computer language and compatability issues which were foremost in Lick's mind. For at least the prior year, Licklider, who was then running the ARPA IPT Office (then called Command & Control Research), had been pursuing the concept of the "Intergalactic Computer Network," trying to define the problems and benefits resulting from computer networking. In any case, that Hot Springs discussion convinced me that I should change my carrier objectives to concentrate on computer networking and the related communications problems.

One year later, in 1965, a second important meeting took place at MIT. Donald Davies from the National Physical Laboratory in the U.K. was at MIT to give a seminar on time sharing. Licklider, Davies, and I discussed networking and the inadequacy of data communication facilities for both time sharing and networking. Davies reports that shortly after this meeting he was struck with the concept that a store and forward system for very short messages (now called packet switching) was the ideal communication system for interactive systems. His wrote about his ideas in a document entitled "Proposal for Development of a National Communication Service for On-Line Data Processing" which envisioned a communications network using trunk lines from 100K bits/sec in speed to 1.5 megabits/sec (T1), message sizes of 128 bytes and a switch which could handle up to 10,000 messages/sec (Historical note: this took 20 years to accomplish). Then in June 1966, Davies wrote a second internal paper, "Proposal for a Digital Communication Network" in which he coined the word **packet**, a small sub part of the message the user wants to send, and also introduced the concept of an "interface computer" to sit between the user equipment and the packet network. His design also included the concept of a Packet Assembler and Disassembler (PAD) to interface character terminals, today a common element of most packet networks.

As a result of distributing his 1965 paper, Donald Davies was given a copy of an

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1986 ACM-0-89791-176-8-1/86-0051 $00.75

internal Rand report, "On Distributed Communications" by Paul Baran of the Rand Corporation, which had been written in August 1964 (1). Baran's historical paper also described a short message switching network using T1 trunks and a 128 byte message size but was oriented toward providing extremely reliable communications for secure voice and data in a military environment. In all, there were 11 reports written for the Air Force in the Rand Memorandum group, of which a couple were classified and unfortunately the others were very sparcely published in the scientific press, thus their impact on the actual development of packet switching was mainly supportive, not sparking its development - that happened independently at Rand, NPL and ARPA.

The First Network Experiment

Convinced that computer networking was important, the first task was to set up a test environment to determine where the problems were. Thus, in 1966, I set up two computer networks between Lincoln Laboratory's TX-2 computer and System Development Corporation's Q-32 computer using a 1200 bps dial channel (high speed those days). Each computer was operating in time-sharing mode and permitted any program to dial the other computer, log-in, and run programs much as it would execute a subroutine call. The experiment showed that there was no problem getting the computers to talk to each other and use resources on the other computer - time-sharing operating systems made that easy. The real problem uncovered was that dial communications based on the telephone network were too slow and unreliable to be operationally useful. This work, jointly authored with Tom Marill, was published in the AFIPS FJCC proceedings, Nov. 1966 (2). The lesson learned - **a new data communications network is needed in order to successfully network computers.**

ARPANET Development

The chance to develop and build a major computer network experiment based on radically new communications technology came within a few months. I was asked to take over the responsibility of the ARPA Information Processing Techniques (IPT) office and manage and build its programs. ARPA was sponsoring computer research at leading universities and research labs in the U.S. These projects and their computers provided an ideal environment for an experimental network project; consequently, the ARPANET was planned during 1967 with the aid of these researchers to link these projects' computers together. One task was to develop an computer interface protocol acceptable to all 16 research groups. A second task was to design a new communications network technology to support 35 computers at 16 sites with 500,000 packets/day traffic. The initial plan for the ARPANET was published in October 1967 at the ACM Symposium on Operating System Principles in Gatlinburg Tennessee (3). The reasons given at that time for establishing a computer network were:

A. Load Sharing: Send program and data to remote computer to balance load.
B. Message Service: Electronic mail service (mailbox service).
C. Data Sharing: Remote access to data bases.
D. Program Sharing: Send data, program remote, e.g. Supercomputer.
E. Remote Service: Log-in to remote computer, use its programs and data.

The communications network design was that of the now conventional packet network; Interface Message Processors (IMP's) at each node inteconnected by leased telecommunication lines providing a store and forward service on very short messages. The main difference from later packet nets was that the IMP's were located at the computer sites and connected by a short parallel cable rather than a communications line interface.

Also presented at the Gatlinburg Symposium was Donald Davies's first open publication on the NPL packet network concepts presented by Roger Scantlebury, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals." (4) It detailed the concept of a high level packet net with high capacity nodal switches and interface computers in front of mainframe computers. This was the first time that either Davies or I knew anything about the work of each other since our 1965 contact. The NPL paper clearly impacted the ARPANET in several ways. The name "packet" was adopted, much higher speed was selected (50 Kilobit/sec vs. 2.4 Kilobit/sec) for internode lines to reduce delay and generally the NPL analysis helped confirm the concept of packet switching.

Another confirmation of the basic concepts came from finally being able to read the Rand reports on distributed communications. The Rand work was very detailed, since it covered the whole network including microwave and one valuable analysis on routing. Their hot-potato routing algorithum was a useful starting point for the ARPANET routing design.

During 1968, a request for proposal was let for the ARPANET packet switching IMP equipment and the operation of the packet network. The RFP was awarded to Bolt Beranek and Newman Inc. of Cambridge, Massachussetts, in in January 1969. The RFP specified the general packetswitching concept, packet size, and interface protocol so that bidders could not totally change the system concept, to circuit or message switching for example. The two largest computer companies to receive the RFP no bid it because they didnt have mini-computers with which to make an economic bid. BBN bid the Honywell 516 mini-computer which was ideal for the task in 1969. Significant aspects of the network's internal operation, such as routing, flow control, software design, and network control were developed by a BBN team consisting of Frank Heart, Robert Kahn, Severo Ornstein, William Crowther, and David Walden. By December 1969 four nodes of the net had been installed and were operating effectively.

The first set of detailed papers covering the ARPANET were published in May 1970 at the AFIPS SJCC. (5-9) These papers reported the motivation and economics (5), the detailed design of the IMP (6), the network delay analysis and experience (7), the topological design programs and results (8), and the host-to-host protocol (9). These papers showed the world for the first time that **packet switching works**, that it is **economic** and that it is **reliable and virtually error free.** They also provided a complete description of how a working network was designed. As such these papers were the technical and motivational basis for many other network experiments around the world.

The ARPANET utilized minicomputers at every node to be served by the network, interconnected in a fully distributed fashion by 50 KB leased lines. Each minicomputer took blocks of data from the computers and terminals connected to it, subdivided them into 128 byte packets, and added a header specifying destination and source addresses; then, based on a dynamically updated routing table the minicomputer sent the packet over whichever free line was currently the fastest route toward the destination. Upon receiving a packet, the next minicomputer would acknowledge it and repeat the routing process independently. Thus, one important characteristic of the ARPANET was its completely distributed, dynamic routing algorithm on a packet-by-packet basis, based on a continuous evaluation within the network of the least delay paths, considering both line availability and queue lengths.

The technical and operational success of the ARPANET quickly demonstrated to a generally skeptical world that packet switching could be organized to provide an efficient and highly responsive interactive data communications facility. Fears that packets would loop forever and that very large buffer pools would be required were quickly allayed. Since the ARPANET was a public project connecting many major universities and research institutions, the implementation and performance details were widely published. (10, 11, 12, 13, 14). The work of Leonard Kleinrock and his associates at UCLA on the theory and measurement of the ARPANET has been of particular importance in providing a firm theoretical and practical understanding about the performance of packet networks.

The ARPANET was first demonstrated publicly at the first International Conference on Computer Communications (ICCC) in Washington, D.C. in October 1972. Robert Kahn of BBN organized the demonstration installing a complete ARPANET node at the conference hotel, with about 40 active terminals permitting access to dozens of computers all over the U. S. This public demonstration was, for many (if not most) of the ICCC attendees, proof that packet switching really worked. At this time, it was difficult for many experienced professionals to accept the fact that a collection of computers, wideband circuits, and minicomputer switching nodes (equipment totaling well over 100 pieces) could all function together reliably. The ARPANET demonstration lasted for three days and clearly displayed its reliable operation in public. The network provided highly reliable service to thousands of attendees during the entire duration of the conference.

Industry Reception

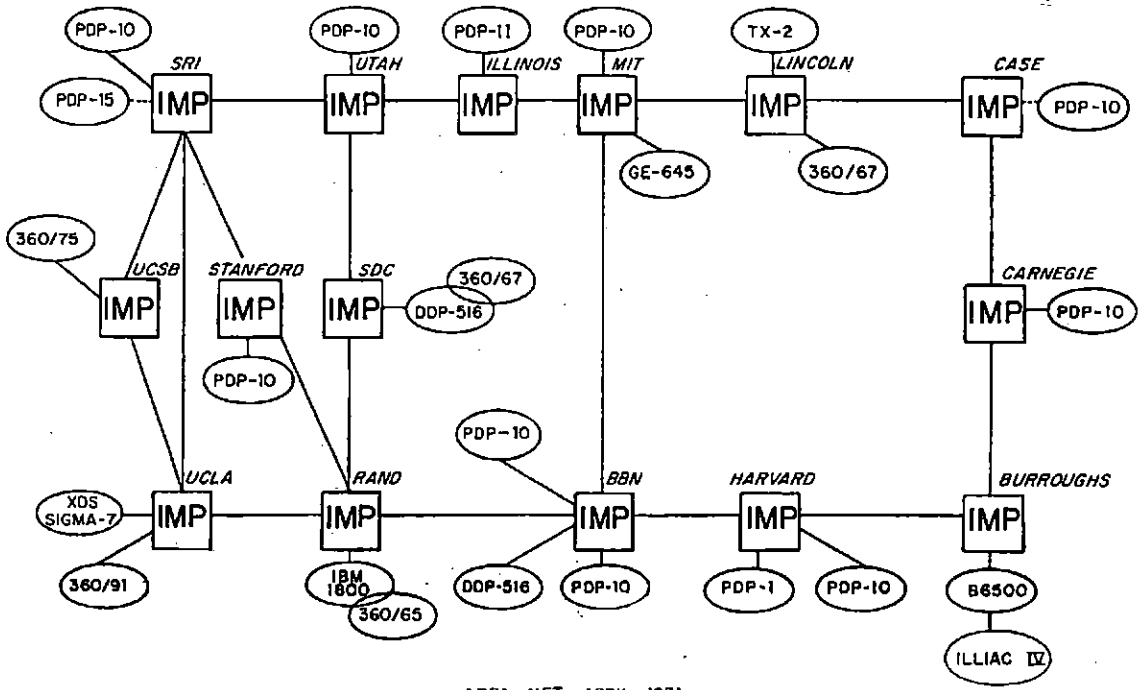
From the first time I distributed a description of packet switching outside the computer research community (the 1967 paper) until about 1975, the communication industry's reaction was generally negative since this was such a radically different approach. In some of the initial technical speeches I gave, communications professionals reacted with considerable anger and hostility, usually saying I did not know what I was talking about since I did not know all their jargon vocabulary. The most common technical flaw suggested (before the ARPANET was built) was that the buffers would quickly and catastrophically run out. After the ARPANET was operating successfully, their pitch changed to be that packet switching would never be economic without the government subsidy. Paul Baran reported the same reaction to his papers when he presented them; this reaction was the major reason his proposals never moved the military. Donald Davies reported a somewhat less angry response from the British Post Office, more one of mild interest but no serious consideration.

I learned a major lesson from that experience; **people hate to change the basic postulates upon which considerable knowledge has been built.** In the case of packet switching, the first postulate to change was the statistical nature of the traffic - data versus voice. The second was that computing was expensive. Some people find it is impossible to consider such a major jolt to their memory organization - they avoid it with putdowns if possible, if not with anger. Other people are more willing to reconsider, but for everyone it requires considerable effort. Those of us proposing packet switching all came from the computing field and did not need to change lots of prior concepts and knowledge. Many of those in the communications field still have not accepted packet switching. (The fight is heating up again as voice packet switching starts to be considered.)

ARPANET Growth

As soon as the first four nodes were brought up and tested in December 1969 the network grew very rapidly. One year later, in December 1970, the network had grown to 10 nodes and 19 host computers. By April 1971, there were 15 nodes with 23 host computers. The topology was as shown in Fig. 1. By this time, it was clear that connecting terminals directly to the network through a PAD-type device was important. Such a device was designed and built in 1970/1971, and the first Terminal Interface Procesor (TIP) was added

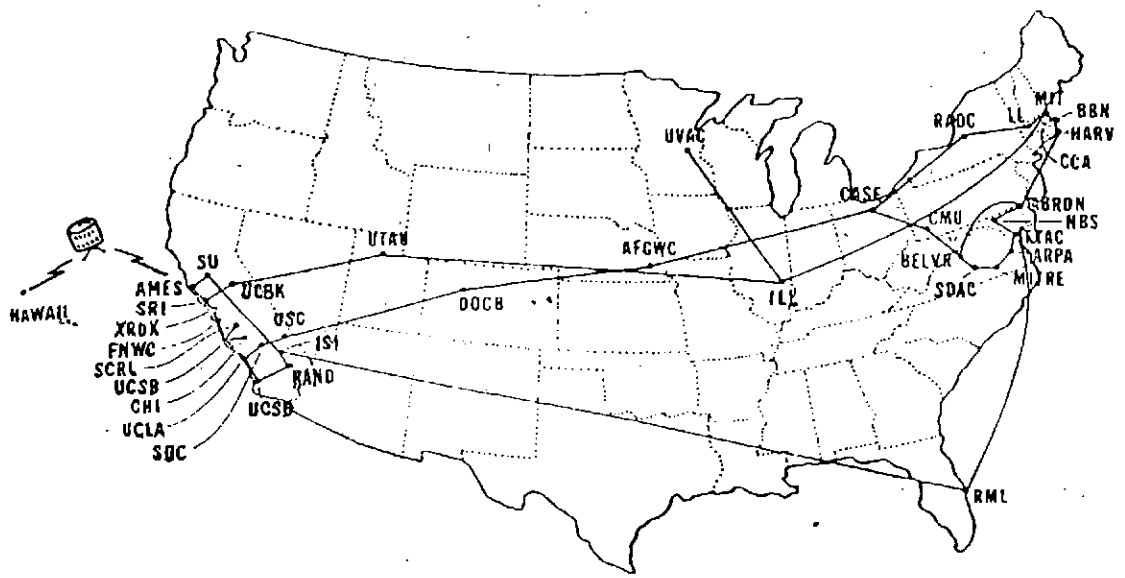
FIGURE 1



ARPA NET, APRIL 1971

FIGURE 2

ARPA NETWORK - JAN 1973



to the network in Aug 1971. This permitted users with no computer to select a computer from all those around the country. In many cases having the user attach his terminal to a TIP and access even his own host(s) through the network was found to be more reliable. This was the start of a trend which today is almost the rule, **workstations should attach to a network, not a computer.** By January 1973, the network had grown to 35 nodes of which 15 were TIP's and was connected to 38 host computers. Fig. 2 shows a map of the network at this point. Network traffic had grown rapidly in 1972, from 100,000 packets/day to 1,000,000 packets/day, exceeding the original estimates I made in 1967. Also in 1973, the first satellite link was added to the network with a TIP in Hawaii; later in the year a pair of TIP's were added in Europe. In September 1973, the network was up to 40 nodes, 45 host computers with internode traffic of 2,900,000 packets/day, had clearly reached a stage of operational stability, heavy usage, and was by any measure a major success. It was at this point that I left ARPA to spread the technology to the commerical world.

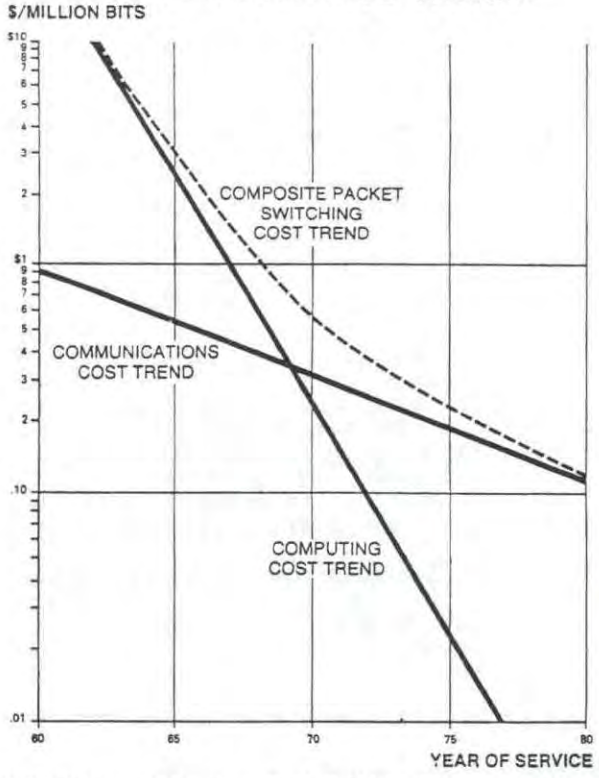
The ARPANET has continued to grow since 1973, with 111 host computers in 1977 and more than 400 hosts in all the interconnected networks by 1983. The network has become a utility for both ARPA and the Department of Defense as a whole. Research has continued into internetting and packet radio but little change has occurred in the basic network.

1969 Crossover

In 1974, after it was clear that the technology worked, I finally spent time analyzing the economic trends which made packet switching possible.(15) These trends could have been analyzed in the mid-sixties if someone were to have asked the right questions, but as it often turns out they were examined after the event they could have predicted. Simply, since packet switching requires more computation than circuit switching but saves transmission band- width, the trend analysis looked at the cost of computation and the cost of communications and found the situation illustrated in Fig. 3. For packet switching the cost of computation dominated in the early 1960's but in 1969 the cost curves crossed and afterward the cost of communications dominated. The composite cost of packet switching thus fell below the cost of circuit switching also about 1969 and since then the margin of advantage has quickly widened to where it equals the full peak to average transmission ratio for data of about 8 to 1. Subsequently, in a 1982 paper (16) I did a similar analysis for voice and concluded the crossover there was in 1978 and therefore packet voice would have been cheaper. In a somewhat similar fashion to data, the industry may recognize this ten to fifteen years after the event.

FIGURE 3

PACKET SWITCHING:
COST PERFORMANCE TRENDS



INCREMENTAL COST OF MOVING 1M BITS (1 KILOPACKET) ON A NATIONWIDE NETWORK

Impact on Computer Resources

One of the original goals of the ARPANET was resource sharing to optimize computer resources. In 1973 I surveyed the ARPANET users to see what alternative computer facilities they would need if they did not have the network. In a number of cases there were ARPA projects which did not have a computer at all but utilized computer power from resource centers which had developed. They obtained access through a TIP, typically onsite, and used one of a collection of computers often accross the country. They were generally happy not to have to run their own computer and had far more reliable service than if they had been limited to one machine. Also due to the statistics of large groups and the benefit of multiple time zones (from Hawaii to Europe) the number of computers required for these users was far less through consolidation. Another group of users were using the Illiac IV remotely for large numerical problems and would have required their own supercomputer or perhaps had to commute if the network were not there. In other cases individuals needed to use software unique to foriegn hardware and thus needed access to other computers besides their own. After finding rational solutions (fiscally sound) to all the ARPA users needs assuming no network, the computing cost was compared to the then actual computer cost and found to be **three times as expensive**. Independent of the details of the survey, it is clear that huge savings were possible (in that era of mainframes) by pooling computing demand. Today when all my computer resources come from micro-computers I couldn't save anything using remote computers for computing power. Supercomputer users still might save but in large part this benifit is mainly historical.

Other Network Development

CYCLADES:In France the interest in computer networks grew quickly during the early 1970's. In 1973 the first hosts were connected to a network called CYCLADES which linked several major computing centers throughout France. (17) The name CYCLADES refers to both the communications subnet and the host computers. The communications subnet by itself is called CIGALE and is a pure **datagram** network, moving only disconnected packets and delivering them in whatever order they arrive without any knoledge or concept of messages, connections or flow control. The ARPANET also operates using datagrams but perhaps the most avid supporter of the concept is the designer of CYCLADES, Louis Pouzin. As with any datagram network, a large part of the communications funtions must be implemented in the host computers; the packet ordering, message formation, flow control, and virtual connections support.

RCP: Another packet network experiment was started in France at about the same time by the French PTT Administration (18). This network, RCP, first became operational in 1974 and was the experimental testbed for the French public network service TRANSPAC. The design of RCP, directed by Remi Despes, differed sharply from that of CIGALE and ARPANET being organized around the concept of the **virtual circuits** rather than datagrams. RCP's character as a prototype public network may have been a strong factor in this difference since a vitrual circuit service is more directly marketable, not requiring substantial modifications to the customers' computer. In any case, RCP pioneered the incorporation of individually flow-controlled virtual circuits into packet switching networks.

Telenet: The success of the ARPANET led Bolt Beranek and Newman (the primary contractor for the ARPANET) to establish a commerical network company, Telenet in late 1972. In October 1973 I joined Telenet as president, and we filed with the FCC to become a carrier and construct a public packet switching network. The FCC approved the request six months later, and Telenet started the world's first public service in August 1975 with 7 nodes. By April 1978 Telenet's network had grown to 187 nodes providing service to 180 host computers and supporting direct terminal access service in 156 cities and interconnections to 14 other countries. Telenet was designed from the start to appear to the user as a **virtual circuit** service with the host interface being implemented over a communications line rather than with a box on site. However, for the first several years Telenet operated a core network based on datagrams copied from the ARPANET but implemented virtual circuits at all interfaces. It wasn't until the complete shift was made to Telenet's TP-4000 packet switch around 1980 that the savings of vitual circuits in the core net could be realized (about 30% for Telenet with a 32 byte average packet size).

EIN: Orginally known as COST Project II and later as the European Informatics Network, EIN is a multi-nation funded European computer network (19,20). The project director was Derek Barber of NPL, one of the original investigators of packet switching in the U.K. It was organized in 1971 but due to the difficulities of multi-national funding it did not become operational until 1976.

Other Public Data Networks: Public packet networks were announced in the U.K., France, Canada, and Japan in the 1973-1974 period. In 1977 the Experimental Packet Switched Service (**EPSS**) in the U.K. and **DATAPAC** in Canada became operational. Also in 1977 a second public network **TYMNET** became approved in the U.S. In 1978 **TRANSPAC** in France and **DX-2** in Japan became operational as public networks. By the early 1980's public packet network service was available in most countries in the world as an international service and as a domestic service in many of these countries.

CCITT Standards- X.25

With five, independent, public packet networks under construction in the 1974-1975 period (U.S.-Telenet, Canada, U.K., France, Japan), there was strong incentive for the

nations to agree on a standard user interface to the networks so that host computers wuld not have unique interfacing jobs in each country. Unlike most standards activities, where there is almost no incentive to compromise and argee, carriers in separate countries can only benefit from the adoption of a standard since it facilitates network interconnection and permits easier user attaachment. To this end the parties concerned undertook a major effort, to agree on the host-network interface during 1975. The result was an agreed protocol, CCITT Recommendation X.25, adopted in March 1976.

The X.25 protocol provides for the interleaving of data blocks for up to 4095 virtual circuits on a single full-duplex leased line interface to the network, including all procedures for call setup and disconnection. A significant feature of this interface, from the carriers' point of view, is the inclusion of independent flow control on each virtual circuit (VC); the flow control enables the network (and the user) to protect itself from congestion and overflow under all circumstances without having to slow down or stop more than one call at a time. In networks like the ARPANET and CYCLADES which do not have this capability, the network must depend on the host (or other networks in interconnect cases) to insure that no user submits more data to the network than the network can handle or deliver. The only defense the network has without individual VC flow control is to shut off the entire host (or internet) interface. This, of course, can be disastrous to the other users communicating with the offending host or network.

Another critical aspect of X.25 is that it defines interface standards for both the host-to-network block transfer and the control of individual VC's. In datagram networks the VC interface is situated in the host computer; there can be, therefore, no network standard for labeling, sequencing, and flow controlling VC's.

The March 1976 agreement on X.25 as the technique for public packet networks marked the begining of the second phase of packet switching: large interconnected public service networks. In the years since X.25 was adopted, many additional packet standards have been agreed on as well. X.28 was adopted as the standard asynchronous terminal interface and X.29, a protocol used with X.25 to specify the packetizing rules for the terminal handler, was adopted as the host control protocol. Also, a standard protocol for interconnecting international networks, X.75 has been adopted.

Packet Switched Voice

Packet networks today typically only utilize trunks and lines up to 64 kilobits/sec in speed. As a result, end to end delay in these networks remains high - over 100 ms. Network delay would however be vastly reduced given high speed trunks (T1-T3) and high speed packet switches (>100,000 packets/sec). Even with a shift to T1 trunks, the network delay would drop to 10 ms plus propagation. With T2 trunks, the delay would not be practically different than with a circuit switch (1-2 ms + propagation). For most data applications this is unnecessary but for voice it is critical. The voice network will often return an echo from analog phones and even 20 ms of delay can make such an echo sound bad. Short packets like 32 bytes (4.5 ms of 56 kb voice) are also necessary to reduce delay. But with high speed trunks and short packets, packetized voice is not only toll quality but also saves 67% of the transmission and switch bandwidth due to silence surpression. Thus in designing tomorrow's central office switch and toll switchs for the telecommunications network it is clearly extremely desirable to use packet switches for voice of any rate (56kb, 32kb, 16kb, etc), for data of all speeds and for switched video. Without packet switching the flexibility is vastly reduced and each service must be switched separately. Packet switches of the required speed are both feasiable and economic today if restricted to fixed-length, short packets (16 or 32 bytes), all flow contol and error checking are done at the network interfaces - not for each link, and a forshortened virtual circuit header is used (about 1 byte) with only the link number. AT&T has a prototype switch of 2 million packets/sec in test and designs exist for switchs up to 35 million packets/sec (21).

The most interesting facet of high speed switch technology is that for large central office or tandem office switches the peak switch throughput is limited by the speed of the fastest memory for a time switch and similarly for a packet switch but the packet switch is able to switch a whole packet not just one byte like the time switch. Thus in general for 32 byte packets, **packet switches can be made about 32 times the maximum throughput of circuit switches**. This becomes very valuable at high level tandem switching offices to switch traffic between many fiber paths.

Based on the importance of coping with variable bitrate demands for data, voice, and video and the cost savings for integrating all services with one switch it is clear that **the whole telecommunications plant will convert to packet switching in the future**. The date is harder to predict but conversion should occur during the 1990's and be complete by 2000. The basic service will be small packets on virtual circuits and each application will have its own interface units. Data services will use an X.25 interface which does the flow control and error checking for the VC on a end-to-end basis.

Future Computer Networks

If we look forward to the year 2000, based on historical trends computers will be about 500 times cheaper per computation than today. Almost all devices in the home, office or plant will have complex computers incorporated into them. Simple local area network facilities like shielded twisted pair will connect the majority of these together and a gateway will connect this local network to the public telecommunication system over

(hopefully a revised and truly integrated) ISDN facility. Distinctions as to which device is the phone may become difficult because many of the device-computers may use the voice network from time to time for warnings, ordering, and certaintly in assisting us to find the party to whom we wish to talk. Phone service today is in great need of automation; we waste large amounts of time non-productively answering and calling to find people. Thus as the telecommunications network brings data and voice together home and office automation will integrate them.

REFERENCES

(1) P. Baran et al, "On Distributed Communications, Vols. I through XI," RAND Corporation Memos, August 1964.

(2) T. Marill and L. G. Roberts, "Toward a Cooperative Network of Time-Shared Computers," Proceedings FJCC 1966, pp. 425-431.

(3) L. G. Roberts, "Multiple Computer Networks and Intercomputer Communication," ACM Symposium on Operating System Principles, October 1967.

(4) D. W. Davies, K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," ACM Symposium on Operating System Principles, October 1967.

(5) L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," Proceedings SJCC 1970, pp. 543-549.

(6) F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," AFIPS Conference Proceedings 36, June 1970, pp. 551-567.

(7) L. Kleinrock, "Analytic and Simulation Methods in Computer Network Design," AFIPS Conference Proceedings May 1970.

(8) H. Frank, I. T. Frisch, W. Chou, "Topological Considerations in the Design of the ARPA Computer Network," AFIPS Conference Proceedings, May 1970.

(9) S. Carr, S. Crocker, V. Cerf, "HOST-HOST Communication Protocol in the ARPA Network," AFIPS Conference Proceedings, May 1970.

(10) S. M. Ornstein, F. E. Heart, W. R. Crowther, S. B. Russell, H. K. Rising, and A. Michel, "The Terminal IMP for the ARPA Computer Network," AFIPS Conference Proceedings 40, June 1972, pp. 243-254.

(11) L. Kleinrock, "Performance Models and Measurement of the ARPA Computer Network," Proceedings of the International Symposium on the Design and Application of Interactive Computer Systems, Grunel University, Uxbridge, England, May 1972.

(12) L. Kleinrock and W. Naylor, "On Measured Behavior of the ARPA Network," AFIPS Conference Proceedings, Vol. 43, NCC, Chicago, Illinois, May 1974, pp. 767-780.

(13) L. Kleinrock and H. Opderbeck, "Throughput in the ARPANET -- Protocols and Measurement," Fourth Data Communications Symposium, Quebec City, Canada, October 1975, pp. 6-1 to 6-11.

(14) H. Frank and W. Chou, "Topological Optimization of Computer Networks," Proceedings of the IEEE, Vol. 60, No. 11, November 1972, pp. 1385-1397.

(15) L. Roberts, "Data by the Packet," IEEE Spectrum, Vol. 11, pp. 46-51, February 1974.

(16) L. Roberts, "Packet Switching Economics," presented at L. M. Ericsson Prize Presentation, Stockholm, Sweden, May 1982.

(17) L. Pouzin, "Presentation and Major Design Aspects of the CYCLADES Network," Third Data Communications Symposium, Tampa, Florida, November 1973, pp. 80-85.

(18) R. F. Despres, "A Packet Network with Graceful Saturated Operation," Proceedings ICCC, Washington, D.C., October 1972, pp. 345-351.

(19) D. L. A. Barber, "The European Computer Network Project," Proceedings ICCC, Washington, D.C., October 1972, pp. 192-200.

(20) D. L. A. Barber, "A European Informatics Network: Achievement and Prospects," Proceedings ICC, Toronto, Canada, August 1976, pp. 44-50.

(21) A. Thomas, J. P. Coudreuse, M. Servel, "Asynchronous Time-Division Techniques: An Experimental Packet Network Intergrating Videocommunication," Proceedings International Switching Symposium, Florence, Italy, May 1984.

Mathematical Laboratories: A New Power for the Physical and Social Sciences

Glen J. Culler

The concept of a mathematical laboratory has been developing throughout the lifetime of computers. The capabilities made available in systems supporting these laboratories range from symbolic integration, differentiation, polynomial and power series manipulation; through mathematical simulation; to direct control experimental systems. About 1961 two trends, one toward what has become known as *on-line* computation, the other toward *time sharing* gained enough recognition to develop national support and subsequently have come to represent what is now known as modern computation. An on-line system provides interactive facilities by which a user can exert deterministic influence over the computation sequence; a time-sharing system provides a means by which partial computations on several different problems may be interleaved in time and share facilities according to predetermined sharing algorithms. For reasons of economy it is hard to put a single user in direct personal control (on-line, that is) of a large scale computer. It is equally or more difficult to get adequate computation power for significant scientific applications out of any small scale economical computer. Consequently, on-line computing has come to depend upon time-sharing as its justifiable mode of implementation. On the other hand, valuable on-line applications have formed one of the major reasons for pushing forward the development of time-sharing systems. At present, both efforts have reached such a stage of fruition that one finds many systems incorporating selective aspects of the early experimental systems of both types.

In this chapter, we will bring out some of the key features of highly interactive direct control systems that have implications for continuing design efforts aimed at furthering the development of experimental mathematical laboratories. We then present a brief description of the foundations of an existing facility at the University of California at Santa Barbara and illustrate its use in a typical application. Finally, we discuss some extension of the system which will provide deeper power for future experimental applications.

Reprinted from Interactive Systems for Experimental Applied Mathematics, Proceedings of an ACM Symposium. Edited by Melvin Klerer and Juris Reinfelds. © 1968, Association for Computing Machinery, Inc. Published by Academic Press, Inc.

1. The Operation-Operand Orientation for Direct Control

A primary aspect of modern computation in contrast to early computation is that of direct user control of computational processes. This is sometimes confused with very early "open shop" computing, in which the user was his own operator, but actually there is no similarity. The fact that a user in direct control commands his own console is only superficially analogous to his being his own operator: the important distinction lies in the program power for direct control associated with his console. This is relatively new and has required some software (computer programming) sophistication over and beyond that of "assemblers" and "compilers". At present, the logical structure required for direct control is well understood and can be simply described. Consider, first some examples of direct control with which we are all familiar; for example, a musician playing an instrument, a person driving an automobile, an operator running a crane, and a child on his skateboard (or surfboard, for that matter). In each instance, there is a lively dynamic situation in which the person is actively involved as the guiding force. A certain amount of prior training is required for success and the variance in training and skill over all people is pronounced. Also, in each case, thoughtful designers have attempted to improve the performance capabilities through the nature of the controls provided. But, no matter how well designed a piano may be, it is not possible for all people to play equally well. It may be the case that, the greater the control facilities, the larger the deviation in final performance. Thus, the essence of design quality should lie in the ultimate performance capabilities under talented control rather than in the elementary creature convenience for the initial learning period of the training. This is not to say that learning difficulties should be ignored, but that ultimate power should not be sacrificed for early convenience.

To design a direct control system, we first isolate the controlling elements (the operators) and their effects, (the transformation of the operands) and then provide a means for their interactive combination. For the design to be really successful, a deep understanding of the nature of the users and their performance desires is required. In the context of human involvement, the existence of Stradivarius violins and the yearly change of automotive design attest to the difficulty of the total direct control problem.

Abstracting from the examples above, we see that a logical structure for direct control consists of a set of operations together with a set of appropriate operands on which the user (a performer, driver, or athlete) can determine which operators

are to be applied to which operands during dynamic, real-time activity. If there, are no operators that make it possible to concatenate sequences of other operators, the system is said to be manual; when there are concatenation operators, the system is said to be programmable and a program is precisely a concatenation of operators with specified operands. If the operands must be absolutely specified at the time the concatenation is formed (i.e., when the program is constructed) then the system is absolutely programmable; if the operands may be specified by variables, then it is variably programmable; if the operators may be specified by variables, then it is substitutionally programmable.

As regards the selection of definitions of particular operators for a given application, very little can be generally said beyond the statement that the designer should try very hard to isolate the most maneuverable set of operators as primitives; this tends to shorten and simplify the programs obtained through concatenation. But there are some observations we can make pertinent to the selection of operators that are independent of the applications area.

RESET

> If there is a concatenation operator, then there should also be a reset operator which can override all other operators and force a stable halt; thus returning the system to manual control.

TEST

> There should be some from of logical operation that permits a conditional selection of a successor (in computer terminology this is called a brance instruction).

ENTER

> There should be some form of communication operator whereby the user can insert manual responses to program requests without destroying the requesting program's control.

REPEAT

> A fixed sequence of operations should be able to be recalled and performed any specified number of times.

Of course, we do not suggest that there is only one way of defining and utilizing the above facilities. Indeed, they can be very simple and minimal in capability or elaborate and extensive according to taste; but if the system does not have at least a minimal form of each of the above, then if suffers substantial restriction of its ultimate power.

Concerning concatenation operators, we have found it convenient to include several types within the same system. The reason for this is that with distinguished concatenations we are able to provide embeddings of program control that enrich the logical structure of the system. We'll return to illustrate this point later when we discuss console programming.

As a means to give specific explanation of how direct control systems can be used in mathematical laboratories, we now formulate an operational calculus in the area of discrete analysis and then detail its implementation in a direct control system. Following this, we will show its application in a laboratory system for the study of speech wave forms and finish the chapter with a brief description of the extension to more complicated data structures and associated operations.

2. An Operational Calculus for Discrete Analysis

The set of operations defined below have been chosen for their simplicity and ease of combination and have with-stood the test of experimental usage for several years and in several systems. The operands include single numbers, lists of numbers (one-dimensional arrays), double lists of number (two-dimensional arrays) and matrices; all of these are allowed to be either real or complex. Since the operands are of several categories, we must include a means for specifying the category. One way to do this is to set aside special variables for each category as illustrated by the expression:

$$f(x) = \sum_{j=0}^{\infty} \left(\sum_{k=1}^{j} A_{jk}\, e^{-kx} \right) \frac{x^j}{j!}$$

which shows the almost unwritten but broadly accepted rules of mathematical notation. THis may well be the best approach when the range of categories is properly restricted and when the conventions of a particular problem area are adhered to. But when many application areas utilize the same system each with their own notational habits, then this approach becomes unwieldy, though certainly not impossible. We select another approach which has been quite successful in practice. This consists of explicitly declaring the category; that is, of providing category operators:

REAL	Operands shall be real.
CPLX	Operands shall be complex.
I	Operands denote single numbwers.
II	Operands denote one-dimensional lists of numbers.
III	Operands denote two-dimensional lists of numbers.
IV	Operands denote matrices.

The operators are defined in relation to the operands as detailed below; they can be grouped as local and global by the nature of their definitions; local operators can be further grouped as arithmetic or as elementary functions. The main content of the definitions is best illustrated for II REAL; that is, for operators on one-dimensional real arrays. Thus in the definitions below, let each letter A,...Z denote a list of n components

$$A = (a_1, a_2, a_3, ..., a_n)$$

such that each component [figure] n is substituted by a real number prior to any actual computation.

A. Operator Definitions for II REAL

1. Local operators are those which operate local to a coordinate index. This means increasing or decreasing the number of coordinates would not essentially change the consequence of these operations at earlier coordinate positions.

a. The arithmetic operators consist of $\oplus$, $\ominus$, $\odot$, $\oslash$, SQ, SQRT, NEG, INV, DIFF, SUM, MOD:

$X \oplus Y = (x_k + y_k \mid k = 1, \ldots, n)$

$X \ominus Y = (x_k - y_k \mid k = 1, \ldots, n)$

$X \odot Y = (x_k \cdot y_k \mid k = 1, \ldots, n)$

$X \oslash Y = (x_k / y_k$, if $y_k \neq 0$; 0, if $y_k = 0 \mid k = 1, \ldots, n)$

SQ $Y = (y_k^2 \mid k = 1, \ldots, n)$

SQRT $Y = (\sqrt{y_k}$, if $y_k > 0$; 0, if $y_k \leq 0 \mid k = 1, \ldots, n)$

NEG $Y = (-y_k \mid k = 1, \ldots, n)$

INV $Y = (1 \oslash Y)$

DIFF $Y = (y_{k+1} - y_k \mid k = 1, \ldots, n-1; 2y_n - 3y_{n-1} + y_{n-2} \mid k = n$

SUM $Y = (\sum_{j=1}^{k} y_j \mid k = 1, \ldots, n)$

MOD $Y = (|y_k| \mid k = 1, \ldots, n)$

b. The elementary function operators consist of SIN, COS, LOG, EXP, ATAN, ARG, DEL.

SIN $Y = (\text{SIN } y_k \mid k = 1, \ldots, n)$

COS $Y = (\text{COS } y_k \mid k = 1, \ldots, n)$

LOG $Y = (\ln y_k$, if $y_k > 0$; 0, if $y_k \leq 0 \mid k = 1, \ldots, n)$

EXP $Y = (e^{y_k} \mid k = 1, \ldots, n)$

ATAN $Y = (\text{TAN}^{-1} y_k \mid k = 1, \ldots, n)$

ARG $Y = (\pi$, if $y_k < 0$; 0, if $y_k \geq 0 \mid k = 1, \ldots, n)$

DEL $Y = (1$, if $y_k = 0$ or if $y_k y_{k+1} < 0$ and $|y_k| \leq |y_{k+1}|$ or if $y_k y_{k-1} < 0$ and $|y_k| < |y_{k-1}|$; 0, if otherwise $\mid k = 1, \ldots, n)$

2. Global operators are those which do not operate local to coordinate indices and are those for which increasing or decreasing the number of coordinates would effectively alter the consequences of the operation at earlier coordinate positions. These are LS, RS, REFL, MAX, ID, EVAL;

$$\text{LS } Y = (y_2, y_3, \ldots, y_n, y_1)$$

$$\text{RS } Y = (y_n, y_1, y_2, y_3, \ldots, y_{n-1})$$

$$\text{REFL } Y = (y_n, y_{n-1}, \ldots, y_2, y_1)$$

$$\text{MAX } Y = \left(\max_{j=1,\ldots,n} y_j \;\middle|\; k = 1, \ldots, n \right)$$

$$\text{ID } Y = \left(\frac{2k - n - 1}{n-1} \;\middle|\; k = 1, 2, \ldots, n \right)$$

$$(X, Y) \text{ EVAL } U = (U, Y(U)) \text{ to first order.}$$

This later operation requires further specification. First extend X by augmenting with $x_0 = -\infty$ and $x_{n+1} = +\infty$

$$-\infty \le t \le \min_{k=1,\ldots,n} X; \quad Y(t) = Y(x_{min})$$

$$\max_{k=1,\ldots,n} X \le t \le \infty; \quad Y(t) = Y(x_{max})$$

Finally, for each value of U, (i.e., u_j, j=1, ...m; m not necessarily equal to n) determine x_μ and x_ν such that

$$x_\mu = \left(\max_{k=0,\ldots,n+1} | \; x_k \le u_j \right)$$

$$x_\nu = \left(\min_{k=0,\ldots,n+1} | \; x_k > u_j \right)$$

and choose

$$y_j = y_\mu + \frac{u_j - x_\mu}{x_\nu - x_\mu} (y_\nu - y_\mu).$$

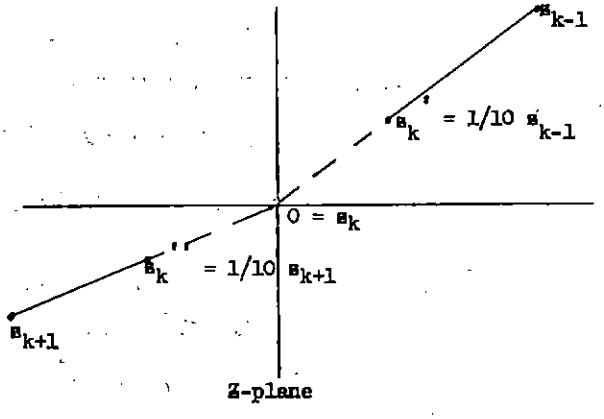
In the exceptional case when some of the u_j's fall outside the range of X, the extension of Y given above gives Y one constant value to the left of x_{min} and another constant value to the right of x_{max}. Also, notice that no ordering of x-values or the 8-values has been assumed; this is important for the required generality of the EVAL operator. As defined, it provides a basis for graphical composition of functions and makes it possible to change discrete variables in the middle or a calculation. When some of the functions occurring in a computation are only known discretely and not at the same intervals used in the calculation, then some form of this EVAL operator is mandatory.

B. Extension of II CPLX

The extension of II REAL operators defined above to the II CPLX operators is accomplished by first introducing complex lists:

$$Z = X + iY; \qquad s_k = x_k + iy_k, \; k = 1, \ldots, n$$

and then defining the arithmetic operators using complex arithmetic on corresponding coordinates. All this is immediate up to the selection of branch cuts as required in the definition of LOG and SQRT. The best way we have found for maintaining compositional compatibility is to define ARG with the facility of branch specification and define all other functions requiring branch definitions in terms of ARG. If we think of a one-dimensional complex array as an ordered set of vertices of a polygonal arc in the complex plane, then the change in argument at adjacent vertices is not more than 2π, consequently, we can computer this change using either the positive or negative principal branch of the argument function. In the event that one of our vertices (Φ_k) actually coincides with the origin, we augment that array and replace that vertex with two new vertices (Φ_k', Φ_k'') chosen on the adjacent polygonal segments..



The final argument function, ARG_+ or ARG. is just the II REAL SUM of the argument difference. Of course, both ARG and MOD of II CPLX are real valued functions, thus part of their calculation consists of making their imaginary parts equal to zero. Also, because conjugation plays such an important role in complex structures, we choose to drop the NEG operator and replace it with *, complex conjugation. Since the real operators when composed turn out to be transformations on Y-coordinates, both NEG and * have the same result: the change in sign of the Y-coordinates.

The global operators also extend easily; LS and RS are direct, REFL brings in a new facility

Let $Z = X + iY$ X, Y, II REAL

$$\text{REFL } Z = \text{REFL } (X + iY) = Y + iX,$$

thus exchanging real and imaginary parts. Coupled with REAL REFL we can still quickly reverse the orderig of a complex array

The extension of MAX results in:

$$\text{MAX } (X + iY) = \text{MAX } X + i \text{ MAX } Y \qquad X,Y \text{ II REAL}$$

The extension of EVAL to II CPLX is not well defined; for it to be so requires strenuous conditions on the arc at which the initial pair is being evaluated, and the operation does not appear to be computationally valuable. The situation is much improved, however, for EVAL III CPLX, which we will discuss later. The operator ID is a fundamental array generator; for II REAL, it produces an interval from -1 to +1 with n equally spaced values; for II CPLX it produces selectively a discrete representation of the unit circle (i.e., the nth roots of unity) or a square about the origin joining (-1-i, 1-i, 1+i, -1+i) with equally spaced points.

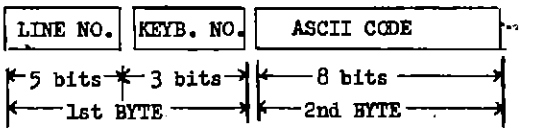
C. Contraction to I REAL and CPLX

Even in problems that are primarily array oriented, we frequently need to use and produce scalars, both real and complex. Consequently, part of our operational calculus consists of operators of category I. The definitions of these operators are obtained by setting n = 1 for category II operators and discarding those that lose their meaning; as for example, SUM, DIFF, MAX, LS, RS, REFL, EVAL, etc.

Starting from this bare frame of operators, we must now show how to accomplish significant control of a computer system and obtain the facilities of a mathematical laboratory.

3. Construction of a Mathematical Laboratory

To build a system that under direct control performs the operations just defined we first introduce a keyboard oriented especially to those operations. Each time one of the keys is depressed a number uniquely corresponding to that key is transmitted to our computer and stored in its memory. The program within the computer analyses this number and determines the type of key just depressed, then responds in accordance with conditions already set up by the antecedents of this key. We make this explicit for our operational calculus. First, observe the keyboard in Figure 1. It is divided into an upper-half and a lower-half; we designate the upper-half as operators and the lower-half as operands. The number transmitted by depressing a key has two parts, the first is the name of the keyboard and the second is the number of the operator key or operand key. These are, of course, binary numbers and are chosen to agree with the American Standard Code for Information Interchange. The format of the number stored in the computer is:



where the line number refers to the current incoming telephone connection is inserted by the equipment servicing the telephone lines, the keyboard number and the ASCII code for the depressed key actually come in on the telephone line. If no telephone lines are used, that is if the keyboard is physically near the computer, then the line number is taken to be zero.

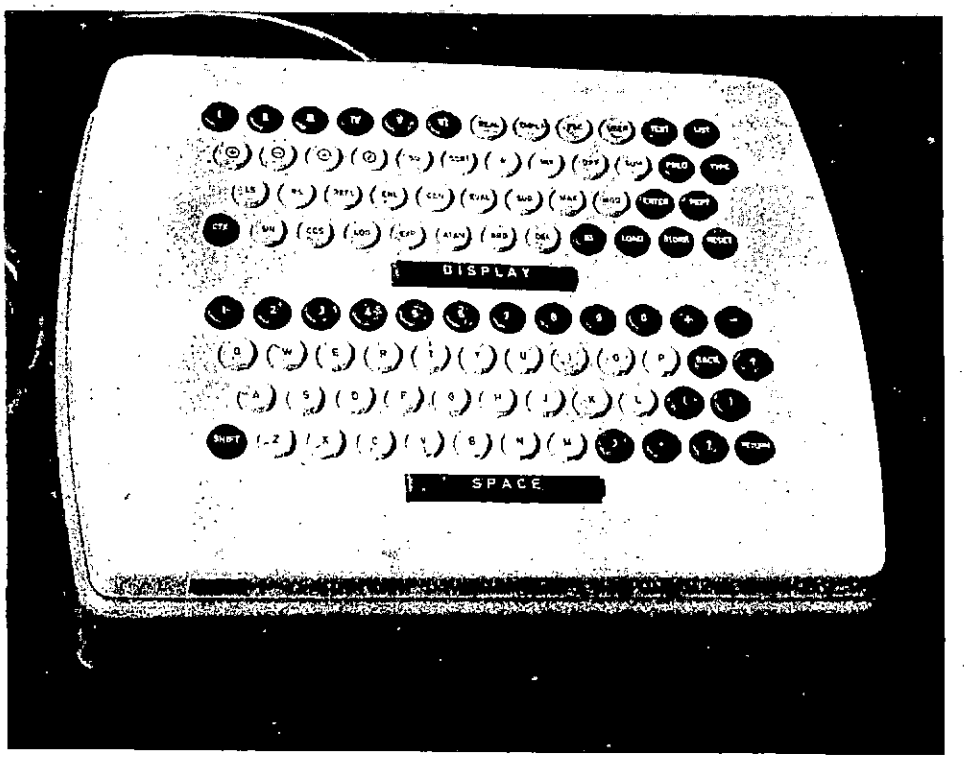


Figure 1. An operational calculus keyboard for discrete analysis.

The operators are grouped according to their system properties and typical use:

A. Operator Category Keys - Blue I, II, III, IV, V, VI.

B. System Change Keys - Yellow REAL, CPLX, SYST, USER.

C. Program Control Keys - Green, Black, Red LIST, TEST, PRED, TYPE, ENTER, REPEAT, DISPLAY, ERASE, LOAD, STORE, RESET.

D. Application Operator Keys - White + - * / SQ SQRT *NEG INV DIFF SUM LS RS REFL ID EVAL MAX MOD SIN COS LOG EXP ATAN ARG DEL.

The operand keys are also grouped according to their usage:

A. Numerical Keys - Blue

B. Alphabetic Keys - White

C. Punctuation Keys - Green

D. Typing Control Keys - Black

Using such a keyboard we can direct the activities of our computer by the sequence of keys depressed. But to do this well, we need to have an accurate mental image of what the computer does in response to each sequence. From the point of view of communication, this is analogous to reading or doing mental arithmetic or in feeling which note should be played next on a musical instrument. The issues involved in bringing this about are threefold:

1. Understanding when an operator-operand sequence is informationally complete,

2. Knowing the family of operand specifications for each operator used,

3. Keeping track mentally of the consequences of each transformation performed.

It is clear that not everybody will do this equally well, but those who use direct control systems with power soon develop these capabilities. A different approach, which is analogous to pushing a bicycle instead of riding it, is to use interactive cues from the computer which force control to go along certain preprogrammed sequences. This is especially useful during the initial training period, but should be curtailed when the user's facility has developed. Systems with too much cueing soon become loggish and stultifying to the able user.

Our method for making the required mental images easy, is to associate a so-called working register with each operator category I, II, III, IV, etc. and picture the consequence of each informationally complete operation as being placed in this working register. As one changes operator categories, the contents of the associated working registers are not changed and hence upon returning to a given category, its contents remain as previously constructed. It is not necessary to prefix each operator with its category designation, that is, the operator category is maintained until a new category is specified. To be more specific about this, we discuss some of the operators in relation to their implied transformations of the working registers. Let the working register be denoted by [figure] and its two associated sub-registers by α and β. To distinguish register categories we use a category subscript:

$$\zeta_I = (\alpha_I, \beta_I)$$

$$\zeta_{II} = (\alpha_{II}, \beta_{II}), \text{ etc.}$$

Now on II REAL, a composition of functions like...

$$f(x) = \log \sin (3x^2 + 1)$$

represented correctly by ...

$$(X, \log \sin (3x^2 + 1))$$

where X is the independent variable running over its prescribed domain. To make such a set of ordered pairs, we construct the discrete domain for X

$$X = (x_1, x_2, ..., x_n)$$

and put in α_{II}, then compose the three transformations and put the result in β_{II}. We must be able to transfer data lists to and from both associated registers α and β of all categories; this is one of the uses of the LOAD and STORE operators.

LOAD:

> Move the data selected by the successor into the working register.

STORE:

> Move the data from the working register into the location defined by the successor.

A typical manner of operation is to load some initial data into B, then to transform it by a sequence of transformations and finally to save it away by storing it as some dependent variable. The point of having LOAD and STORE directly available is to provide specific control over the contents of the working register. On the other hand, there are many instances where the transfers to and from the working registers can be performed implicitly and at a saving of manual labor. Thus, we make implicit transfers also possible where convenient. For II REAL, the operations defined in Section 2 all leave the consequences in β_{II}, which satisfies the compositional conditions noted above. Each operation occurs in a number of situations according to is predecessors and successors, but within the operator sub-groups defined in Section 2A, above, there is sufficient similarity that is adequate for us to illustrate a typical member of each sub-group.

Computations are carried out in complete compositional sequences, that statement of completion being any category key, DISPLAY, STORE, or RETURN. After a computation is performed, the results are left in β, whether or not they have been saved by the calculation. A new compositional sequence can be initiated by an operator or operand; if initiated by a variable operand which is not immediately followed by =, then an implicit LOAD is effected; if the variable operand is immediately followed by =, then the composition sequence will end by effecting an implicit STORE associated with that variable.

To illustrate the above relations of compositional sequences, we analyses equivalent sequences of computing F = log sin $(3x^2 + 1)$. Assume that we are using II REAL and that X has previously been constructed.

A. LOAD X SQ * 3 + 1 SIN LOG STORE F

B. $3X^2$ + 1 SIN LOG = F

C. LOG SIN $(3X^2 + 1)$ = F

D. F = LOG SIN $(3X^2 + 1)$

These all make the results of computing:

$$F_k = \text{LOG SIN } (3X^2 + 1)$$

available as the discrete data for the variable F, but they show different aspects of the direct control system. In a, the composition sequence is constructively ordered but greatly different in appearance from the original expression. After each step of A the contents of β_{II} are explicitly known to the user:

LOAD X	$(x_1, \ldots, x_n) \in \beta_{II}$
SQ	$(x_1^2, \ldots, x_n^2) \in \beta_{II}$
⊙ 3	$(3x_1^2, \ldots, 3x_n^2) \in \beta_{II}$
⊕ 1	$(3x_1^2 + 1, \ldots, 3x_n^2 + 1) \in \beta_{II}$
SIN	$(\sin(3x_1^2 + 1), \ldots, \sin(3x_n^2 + 1)) \in \beta_{II}$
LOG	$(\log \sin(3x_1^2 + 1), \ldots, \log \sin(3x_n^2 + 1)) \in \beta_{II}$
STORE F	$(\log \sin(3x_1^2 + 1), \ldots, \log \sin(3x_n^2 + 1)) \in \beta_{II}$

In B, we begin with an implicit LOAD, use the power operator ↑ which does not appear except in forming the superscript, and replace STORE with which is equivalent in this context. In C, we use an interpretive ordering of the operators and obtain a familiar appearance as a formula but lose any apparent correspondence to the transformations performed. In D, we precisely have mathematical notation and are satisfied to be sure that F will end up with the right answers. In such simple circumstances it, of course, makes no real difference which approach is used. In practice, it depends upon the experimental requirements of the situation; that is, if there are no possible computational difficulties and all the compositional sub-sequences are unimportant to the user, then the automatic formula approach of D is most satisfying; but if something has gone sour, (say through singularities, bad distribution of data, or loss of accuracy through numerical pathologies) then it is important to be able to construct the results through step by step transformations of β_{II} and A is the most satisfying. Since both circumstances occur frequently in using a mathematical laboratory, we must provide both facilities (B and C are irrelevant, they are simple variations of A and D).

The final ingredient of the laboratory, is a display tube which permits visual understanding of the operations carried out by use of the DISPLAY operator and will be illustrated in the next section.



Figure 2. User consoles in the University of California, Santa Barbara mathematical laboratory.

4. Experimental Analysis of Speech Wave Forms

In a total system like that outlined above, there are so many operations each with several instances and special uses that we cannot present all contingencies and facilities short of producing a fairly large user's manual. But we can convey some sense of a user's experience to the reader by tracing through and explaining a brief but fairly typical actual sequence of operations as they occurred when we were studying how to analyze wave forms arising in natural speech. The equipment used consisted of an early mathematical laboratory built around the RW-400 polymorphic computer with the addition of an IBM 1800 process control computer. To the latter computer we attached a microphone input and an audio speaker output as well as a two-way data transfer connection to the first computer. Following the operator-operand philosophy outlined in Section 2, we selected a set of primitives that gave us control of the microphone input and the speaker output and programmed them for the IBM 1800. The control of these operations was implemented as a special category, VII, and added to the former mathematical system. The operations consist of controls for recording levels, for selecting which sub-lists of data to listen to, for selecting and transferring data to and from a back-up disk. Thus, we can digitally record live speech, bring it into the mathematically oriented system and decompose it by whatever means occur to us. In the other direction, we can manufacture mathematical functions and use them to build data lists in the IBM 1800; then make the 1800 output this list through an audio speaker for us to hear.

The data buffer, αVII, consists of 7440 memory locations subdivided into 60 blocks of 124 numbers each. We chose to use some letters on the alpha-numeric keyboard followed by specification of blocks and position within the blocks as non-numerical operations.

A. Category VII Operators

RETURN:

Initiate operation just specified.

R k:

Begin recording input data when the signal exceeds the threshold k (30 threshold levels are available).

Q:

Quiet, cease output to the speaker.

L a, b, c:

Listen; initiate output to the speaker starting with block a, position c, and output b consecutive blocks before delay.

D a:

After each speaker output delay a times a unit interval before repeating the output.

I a,c:

Input 1 block starting at block a, position c and place the data in β_{II} of the RW-400 system.

O a,,c:

Output 1 block from β_{II} of the RW-400 system and put it in block a starting at location c.

S k:

Apply a k-point smoothing formula to that part of the data list in α_{VII} addressed by the last L a,b,c operator.

LOAD k:

Transfer the data on disk cylinder K into α_{VII} of the 1800.

STORE k:

Transfer the data list in α_{VII} of the 1800.

LS:

Specify a variable to be used as a vector of scales for the blocks of α_{VII}.

RS:

Rescale all blocks to a common scale.

B. Experimental Analysis of SAT

The pictures below showing a grid were taken with a Polaroid camera attached to a Tektronix storage oscilloscope that records the analog voltage signal output from the IBM 1800. The others not showing a grid were taken with the same camera attached to a second such oscilloscope used as a display device for alpha-numeric and graphical output from the RW-400.

To initiate a run, the user presses

RESET LOAD USER

and receives the computer's response requesting his user number. After the number is provided, the computer retrieves his complex of user programs and accumulated data and types out the user's own system message.

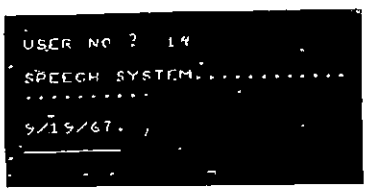


Figure 3. Start of a run

To record a word for study, we pressed

VII R 7 RETURN

and pronounced "sat" into the microphone. After adjusting the oscilloscope scales as shown below the graph, we see the complete wave form of the recorded word.

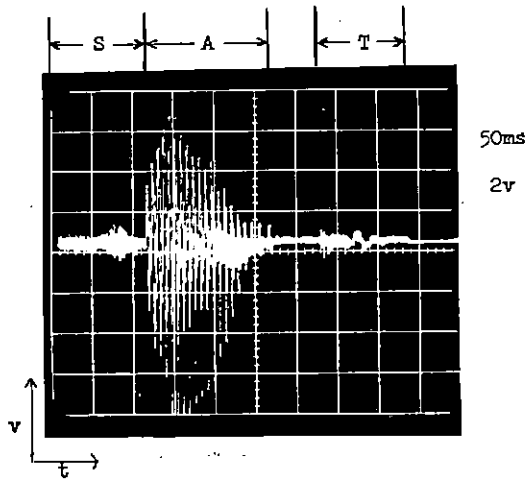


Figure 4. Wave form of SAT

In order to view the character of the wave form producing S we pressed

L 1, 17 RETURN

and adjusted as is shown. Since the oscillations were too rapid to see we then pressed

L 9, 1 RETURN

which allowed us to see block 9 alone, where the large amplitude of S occurred. Compressing the time scale resulted, finally, in a good picture of the mixture of wave functions in S.

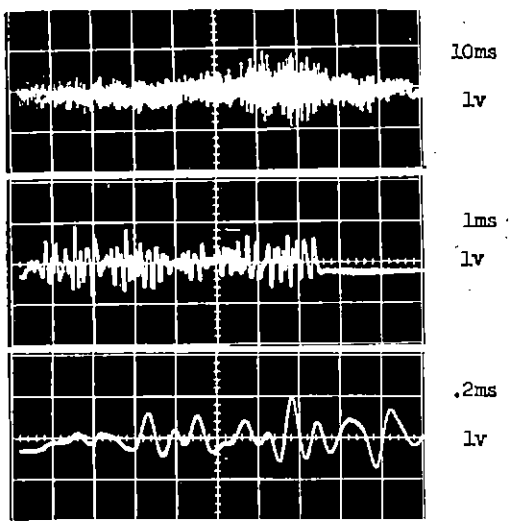


Figure 5. Details of the S wave form

Carefully observing the form of the last graph presented above allows one to identify the actual ingredients of S as a sequence of Gaussian wave functions, i.e. normalized derivatives of

$$EXP((-t^2/2\sigma^2)$$

having an assortment of k-values (the order of the derivatives) and occurring haphazardly in time. This decomposition of speech wave forms into Gaussian wave function components is a new approach to the classification, analysis and synthesis

of speech elements and, of course, cannot be properly treated here. But we can illustrate a simple instance of it. For this purpose, we will study the data producing the A in SAT.

The three pictures of Figure 6 were obtained respectively by

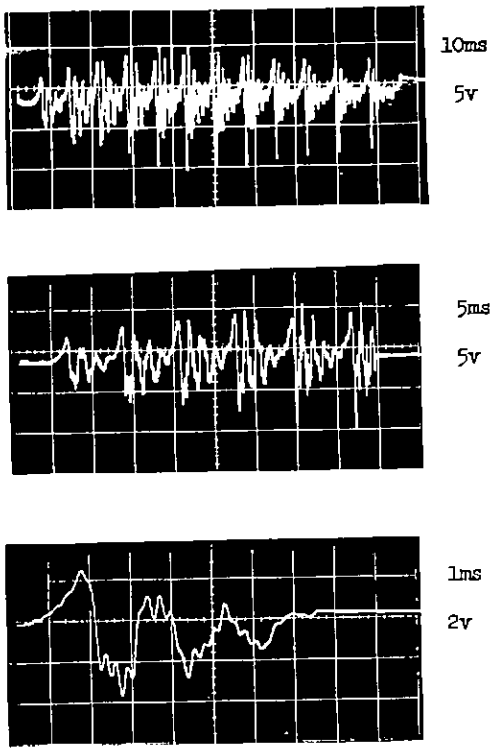


Figure 6. Pitch periods of the A in SAT

L 14, 14 RETURN

L 14, 6 RETURN

L14, 1, 70 RETURN

and show a successive blow-up of the initial pitch period of the A (the wave form lying between the larger extremes in a nearly recurrent manner is called a "pitch period"). From the first of these graphs, we can observe a relative rise in the second and third positive extremes. The fourth pitch period seems to be fairly representative of this family, so we all it into the mathematical system by pressing

I 18,,50

still of category VII, then enter category II REAL for the ensuing analysis, as shown in Figure 7, we then put the curve on maximum scale by adding zero,

\+ 0

and then save it as the variable operand Q and finally display the curve. Each such pitch period is made up of essentially three signals distinguished by their oscillations frequencies. There are many ways to attempt to separate out these sub-signals; we shall use the DIFF operator in combination with an integral mean smoothing operator as in VII S.K above, or a five point smoothing operator as shown in Figure 8. The graph in Figure 8 is the result of applying two successive smoothings. Subtracting this smooth function gives a first approximation to the high format.

Due to inherent properties of smoothing operators, we cannot be sure that this approximate formant is pure; that is, it may still contain some components of the two low formants. Running it through the same smoothing process results in a function of small amplitude.

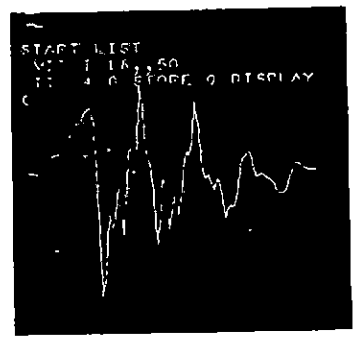


Figure 7. The fourth pitch period of A in SAT

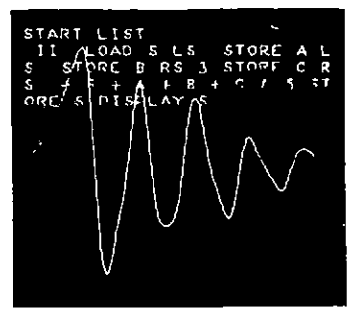


Figure 8. Combination of Low Formants

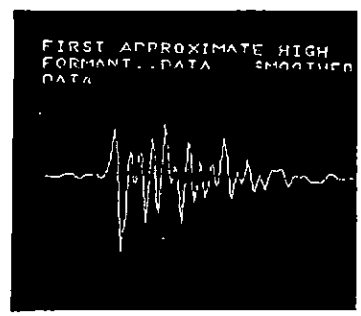


Figure 9. Approximate High Formant

When this function is subtracted from that shown in Figure 9, we obtain the extracted high formant as shown in Figure 10.

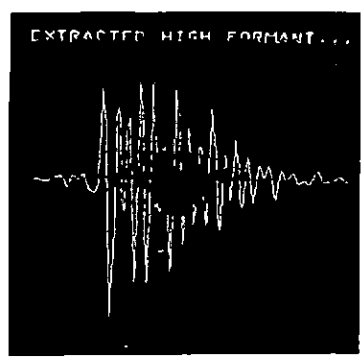


Figure 10. Second approximation to the high formant

Another method based on the DIFF operator uses the fact that for well separated frequencies, and fairly close data points, the difference operator separates the relative amplitudes of the formant. Applying DIFF DIFF NEG to the original data curve of Figure 7, produces the graph shown in Figure 11. The normalization of this function is lost in the DIFF process, so only shapes should be compared.

Returning to the consideration of the low formants, we can use DIFF again to try to split apart this pair. The dotted graph in FIgure 12 is the smoothed graph above, and the solid graph is its DIFF. Notice particularly the disfiguration of the solid graph near the large positive maximum, this shows where two wave functions are overlapping.

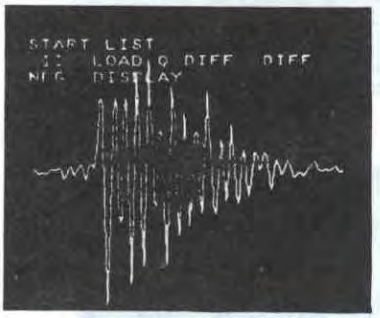


Figure 11. Unnormalized approximate high formant using the DIFF DIFF operator

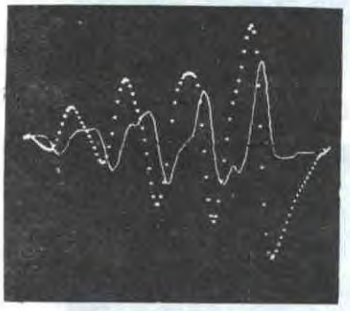


Figure 12. DIFF applied to the combined low formants

Our approach now is to identify the wave functions in this difference graph and compose them to approximate it, then to use the SUM operator to obtain an approximation of the dotted graph of Figure 12. Experimenting with the Gaussians width results in the choice of

$$1/2\sigma^2 = 81$$

as one which looks hopeful. To see this compare figure 13 with the shapes of the extremes in Figure 12. Our conjecture is that for this A sound, the deep first minimum results from the second derivative of the Gaussian and the second wave function occurring later is just its normalized derivative. Using the LS and RS operators we position the minimum and with NEG, MAX, INV we choose amplitudes and thus construct the wave functions shown in dotted curves of the first two graphs of Figure 14. Adding the resulting wave functions together, we next apply DIFF and then reposition and renormalize the result to fit the second pair of extremes as shown in the dotted graph of the third picture. Continuing this process once more then completes our approximation and using SUM results in the dotted graph of Figure 15, which is compared to the solid curve resulting from the smoothing operator. Now our approximation has no components of the lowest formant in it which accounts for the changes in elevation on the right hand side of Figure 15. Finally, adding in the high formants allows comparison of the original data curve with our fit.



Figure 13. The first three Gaussian wave functions

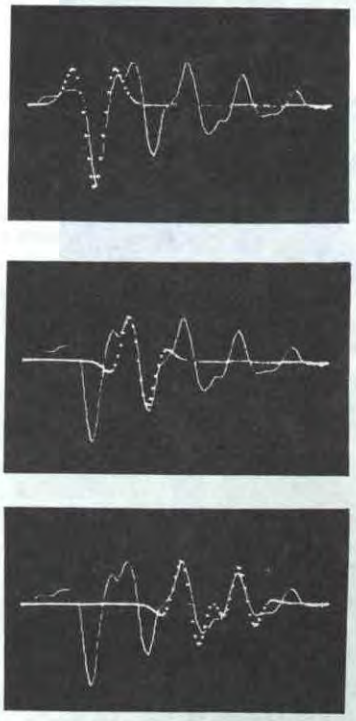


Figure 14. Fitting the difference curve by wave functions

If one looks carefully back at the extracted high formant, he will see one of its constituent wave functions. The procedure just outlined will provide a representation in the same manner and result in a total wave function representation of the original data curve.

Going back to the speech computer, we use

VII L 40. 19 RETURN

and get a good view of the wave making the T sound. Since we are always able to see and hear the data simultaneously, the identification of the linguistically effective parts of the wave forms is easily made. The data for T is similar to that

for S except for the single large slow wave function clearly evident in the T.

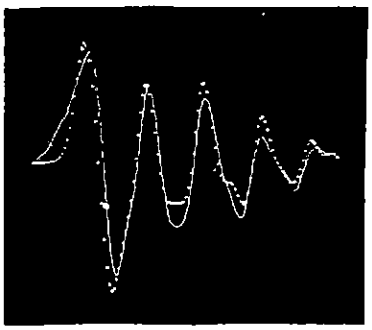


Figure 15. Comparison of smoothed data and wave function fit

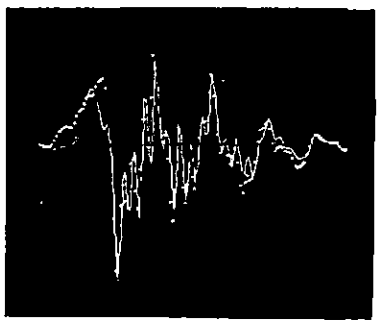


Figure 16. Comparison of original data and wave function fit

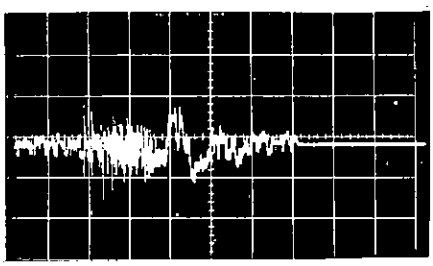


Figure 17. The wave function for the T in SAT

5. Extension or Higher Category Operations

The requirements of a structure supporting operations of categories II and IV both REAL and CPLX can be made clear by formulating several problems that point the way towards generalized data objects and their associated transformations.

A. Problems for Category III REAL

1. Integral equation

Let s, t be independent variables running over an interval [a,b] and F (s,t,u) a function of three variables such that for each point (s,t) ε $[a,b]^2$, F is a rational function of U. For some class of non-zero driving functions g(t) we wish to determine U(t) such that

$$U(t) = g(t) + \int_a^b F(s,t,U(s))\ ds.$$

To check whether a given function U(t) satisfies this equation the coefficient functions $A_j(s,t)$, $B_j(s,t)$ must be computed and the rational function of order J constructed.

$$F(s,t,U(s)) = \frac{\sum_{j=0}^{J} A_j(s,t)\ U^j(s)}{\sum_{j=0}^{J} B_j(s,t)\ U^j(s)}$$

If we choose III REAL precisely as a two-dimensional extension of II REAL, and use discrete variables

$$S = (s_1, s_2, \ldots s_n)$$

$$T = (t_1, t_2, \ldots t_n)$$

then the data structure for the coefficient functions

$$A_j(s,t) = (A_j(S_k,t_l) \mid k = 1, \ldots, n;\ l = 1, \ldots, n)$$

are the directly addressable operands for III REAL. The extension of the II REAL operators is achieved by considering the columns of III REAL operands (or rows) to be of the same class (but differently labelled) as II REAL operands. Then II REAL may be used sequentially to build III REAL operands.

2. Properties of surfaces over plane domain

Let H(x,y) be defined for points (x,y) on the closure of a domain D which consists of the interior of a simple closed curve Γ

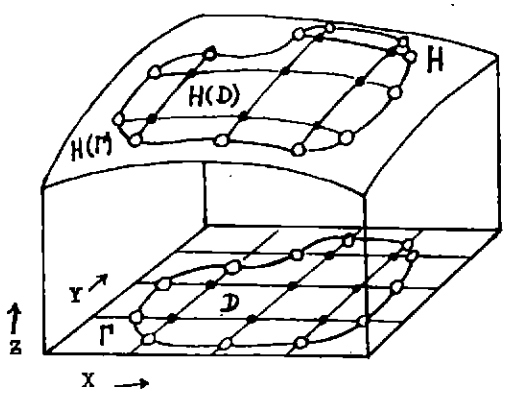


Figure 18. Surface representation of III REAL operands

Since D is not necessarily rectangular we must enrich our data representation by some form of logical selection which permits partial columns and partial rows to be easily used. Now let

$$X = (x_1, \ldots, x_n)$$

$$Y = (y_1, \ldots, y_n)$$

be two REAL operands. An operator W which forms the product set X x Y builds a discrete rectangular domain which is a natural operand for III CPLX, since corresponding to each index pair (j,k) there is a coordinate pair (x_j, y_k) which can be thought of as a

point on a plane grid. Observing that Γ can be represented as a II CPLX operand, we let

X x Y = W, III CMPLX

and

Γ = G, II CMPLX

then on III REAL we generate a real array of 1's and 0's which is the characteristic function of the interior of G with respect to W:

LOAD III CPLX W DEL II CPLX G

where G is insured to be closed by considering the first and last points to be adjacent vertices in a closed polygon. This characteristic function can be used as a logical mask in calculations involving D; also, it is binary by nature, so it can be compressed to bit by bit representation and carried along with W above as the representation of D. As shown in Figure 18 above, it is also convenient to interpolate Γ at the grid line crossings of the W array. This is accomplished on II CPLX by:

LOAD G EVAL III CPLX W

and results in a new representation of Γ consisting precisely of the interpolated intersections of the grid lines of W and the polygon Γ.

From the above, it should be remarked that generalizing DIFF selectively to partial differences and SUM selectively to row sums and column sums, and using ↑ as an "up" shift and | as a "down" shift to add to LS and RS already available, we have appropriate operand and operator representations to work with discrete methods for partial differential equations in two variables; these comments hold both for Marching problems and Relaxation problems. Furthermore, transformations of data representations are easily effected. For example, let:

$Z = (\zeta_1, \zeta_2, \ldots \zeta_l) \in$ II REAL

and

$H = (h_{jk} \mid j=1, \ldots, n;\ k=1, \ldots, m)$

then on III CPLX,

LOAD III REAL H EVAL II REAL Z

results in a complex array consisting of l each arc being a contour of Z = H(x,y) discretely approximated. That is, the above composition of operators constructs a contour map of H at elevations defined by the component of Z.

B. *Problems for III CPLX*

In many respects, the generalization of II CPLX to III CPLX is simpler and more apparent than the extension of II REAL to III REAL. The reason for this rests in the fact that II CPLX already involved plane considerations, and extensions to III CPLX essentially iterates the definitions if II CPLX. Thus, for example, let F be an analytic function over a domain D with boundary Γ. Let $\zeta_1, \zeta_2, \ldots \zeta_m$ be polygonal arcs in D. Choose the III CPLX operand

$Z = (\zeta_1, \zeta_2, \ldots \zeta_l)$

then $F(\zeta) = F(F(\zeta_1), F(\zeta_2), \ldots, F(\zeta_l))$

is the result of applying the "operation" F to Z and is an image arc representation. If D is well spanned by Z, that is, if each point of D is sufficiently near some points of Z, then F(D) is well spanned by $F(\zeta)$ because F was presumed to be analytic on the closure of D. If F is univalent over D, then $F(\zeta)$ is a discreet representation of the conformal mapping defined by F.

As mentioned in Section 2, III CPLX EVAL achieves an appropriate meaning. For example, suppose

$V = (v_1, v_2, \ldots, v_l) \in$ III CPLX

and that each v_j, $j = 1, \ldots, l$ lies in D where D and F are discussed above. The F(v) can be computed by an interpolation on III CPLX:

(Z, F(ζ)) EVAL V

in which each point v_{jk} is located in Z by finding its nearest neighbors, and $F(v_{jk})$ is computed from the function values corresponding to these neighbors.

C. *Considerations for Category IV REAL and CPLX*

There are, of course, a variety of computational mathematical problems that do not fit naturally into an operator-operand format. For these, a well designed mathematical laboratory should be able to exercise library facilities and incorporate any advantageous programs, methods, etc. that exist in conventional systems into a user's own system. There is no reason to try to force an unnatural format on a given problem area. Just as this is "bad" mathematics, so is it "bad" computation. Consequently, it is important to recognize the constructive potentials and natures of mathematical endeavors and select computational approaches in resonance with these.

With this view in mind, we choose IV REAL to have operands that are symmetric matrices and represent these matrices in diagonal form. That is, our matrices are of the form:

M = U * D U

where U is a unitary matrix whose columns are eigenvectors of M and D is the diagonal matrix of eigenvalues of M. The data structure D is like a II REAL operand, and the data structure of U is like a III REAL operand. Now the elementary functions on M obey:

F(M) = U * F(D) U

and hence are computable by II REAL operations. Thus, our operational calculus extends readily to the algebra generated by M.

For IV CPLX, Hermitian matrices follow the above in detail and require no further discussion.

For general matrices, those that are diagonalizable still

permit reasonable structures, but the present stage of experimentation makes us prefer to include general matrices and linear systems in the mathematical library category.

From today's vantage point, it appears that mathematical laboratories have achieved a secure footing; that they provide a number of challenging computational-mathematical-design problems for interested students of modern computer sciences and mathematics.

Acknowledgement

The national government, through the Advanced Research Projects Agency, the National Science Foundation, the Office of Naval Research, and the Rome Air Development Center, have provided support to a number of the experimental mathematically oriented systems listed in the references. While this has been a small segment of the total support provided in computer research efforts, it has been a keystone of the efforts listed here. In several instances the Universities, themselves, and private industry have contributed facilities and hardware and have evidenced a lively amount of interest in these developments. At present, these systems have achieved a stage of utility that makes it important for such support to continue in order to bring a broad range of experimental usage through other campuses not necessarily involved in the early stages of these system's construction.

University of California, Santa Barbara
Santa Barbara, California

References

A. Experimental Mathematical Laboratories

1. The T.R.W. Canoga Park System
   Culler, G.J., and B.D. Fried: An On-Line Computing Center for Scientific Problems, Proc. 1963 Pacific Computer Conference of the IEEE, March 1963, p. 221.
2. The S.T.L. Space Park System
   Culler, G.J., and B.D. Fried: The T.R.W. Two station On-Line Scientific Computer and General Description, Computer Augmentation of Human Reasoning, Spartan Press, 1965.
3. The Columbia University System
   Klerer, M, and J. May: Two-Dimensional Programming, Conference Proceedings of the Fall Joint Computer Conference 1965, Spartan Books, pp. 63-73, 1965.
4. The Amtran System
   Albert, M.R., P.C. Clem, L.A. Flenker, J. Reinfelds, R.N. Seitz, and L.H. Wood: The Amtran Sampler System INstruction Manual, NASA Technical Memorandum, NASA TM X-53342, July 1966.
5. The University of California, Santa Barbara System
   Culler, G.J.: The User's Manual for an On-Line System, On-Line Computing, McGraw Hill, 1967.
6. The MAP System
   Kaplow, R., S. Strong J. Brachett: MAP- A System for On-Line Mathematical Analysis, MIT Publication, MAC TR 24, January 1966.
7. The Harvard System
   Ruyle, A.: The Development of Systems for On-Line Mathematics at Harvard, ACM Symposium on Interactive Systems for Experimental Applied Mathematics, August 1967.

B. User's Publications

1. Culler, G.J., B.D. Fried, R.W. Huff, and J.R. Schrieffer; Solution of the Gap Equation for a Superconductor, Phys. Rev. Letters, vol. 8, p. 399, 1962.
2. Fried, B.D., and G.J. Culler: Plasma Oscillations in an External Electric Field, Phys. Fluids, vol. 6, p. 1128, 1963.
3. Schrieffer, J.R., D.J. Scalapino, and J.W. Wilkins: Effective Tunneling Density of States in Superconductors, Phys. Rev. Letters, vol. 10, p. 336, 1963.
4. Johnson, Kenneth, and Marshall Baker: Quantum Electrodynamics, Phys. Letters, vol. 11, p. 518, 1963.
5. Cheng, Hung, and David Sharp: Formulation and Numerical Solution of Sets of Dynamical Equations for Regge Pole Parameters, Phys. Rev. Letters, vol. 132, p. 1854, 1963.
6. Field, E.C., and B.D. Fried: Solution of Kinetic Equation for an Unstable Plasma in an Electric Field, Phys. Fluids, vol. 7, p. 1937, 1964.
7. Bullock, D.L., : Exchange Ratio in CuF_2 $2H_2O$ TRW/STL Report 9891-6001-RU-000, April 1965.
8. Dixon, W.J., : ΔV to Enter Orbit About Mars, TRW IOC VM-2, April 1965.
9. DeNuzzo, J.: On-Line Solution of 2-D Trajectory Equations, TRW/STL Report 9801-6013-TU-000, May 1965.
10. Nishinago, R.G.: Preliminary Design Considerations for a Gyro-damped Gravity Gradient Satellite, TRW/STL Report 8427-6005-RU-000, May 1965.
11. Margulies, R.S.: Response of a Peak-reading Instrument to a Contaminated Signal, TRW/STL Report 9990-6963-TU-000, June 1965.
12. Pate, N.C., and S.M. Zivi: An Analysis of the Efficiency of Elliots Liquid Metal MHD Energy Conversion Cycle and IT Applicability to the Power Range of 3 to 30 KWE, TRW Systems Report 9806-602-MU-000, July 1965.

13. Fried, B.D., and L.O. Heflinger: Scaling Law for MHD Acceleration, TRW Systems Report 9801-6014-RU-000, July 1965.

14. Fried, R.D., and S.L. Ossakow: The Kinetic Equation for an Unstable Plasma in Parallel Electric and Magnetic Fields, UCLA Report R-3 Plasma Physics Group, November 1965.

15. Fried, B.D. and A.Y. Wong: Stability Limits for Longitudinal Waves in Ion Beam-Plasma Interactions, TRW Systems Report 9801-6015-RU-000, August 1965.

16. Coward, D.J.: . Exact Inversion of Plasma Dispersion Relations, Phys. Fluids.

17. Collins, L: The Calculation of Unpaired Electron Density on the Nucleus of Many-electrons with a Thomas-Fermi-Dirac Potential.

18. Madon, Rabi: Superperturbation and Super-perturbation Theory Bounds on Electron Hydrogen Scattering.

THE AUGMENTED KNOWLEDGE WORKSHOP

Douglas C. Engelbart
MC DONNELL DOUGLAS CORPORATION
Information Systems Group
Cupertino, California
<AUGMENT,101931>

INTRODUCTION

The story of my involvement with on-line workstations begins in early 1951, with a vision and a life-time professional commitment. Over thirty-four years of pursuit have created a lot of personal history, and the object of this historical exercise, the workstation, occupies a unique place in it.

For me, a workstation is the portal into a person's "Augmented Knowledge Workshop" -- the place in which he finds the data and tools with which he does his knowledge work, and through which he collaborates with similarly equipped workers. And further, I consider that the large system of concepts, skills, knowledge, methods, etc. on the human side of the workstation has to be taken into account, in a balanced way, when pursuing increased human effectiveness. So, my workstation-history story embraces a rather large sphere.

The task of writing an historical piece is unfamiliar enough to cause me difficulty by itself, but the associated stirring of old records and old memories has added near overwhelming burden -- dreams, events, people, stresses, pleasures, disappointments, the firsts and the failures. Now, what from all of this -- and how to organize it -- will make an appropriate "history" paper?

I could provide a solid measure of objective reporting -- events and dates, etc. Regarding the general environment that is relevant to the workstation topic, I have been an involved observer of related computer history since 1951. I watched and experienced the supportive hardware, languages and architecture evolve, witnessed the people and efforts that brought timesharing into being, and was even more closely involved with the emergence of computer networks. Through all of this, I was wholly focused on what these things could do for people at workstations. And then there was office automation and personal computers: you don't have to be an old guy to have watched these emerge, but I'm sure they looked different to me than to most.

I could also provide lots of objective reporting about the events and dates associated with the things I have caused or had a direct hand in. There seems to be a lot there that is quite relevant to this "history of the workstation" theme. It was dusty, laborious work, this process of brainstorming for candidates, culling and ordering and trying to describe them in some reasonable sequence and context.

But what I came to realize is that there is one, clearly dominant factor that underlies essentially every cause for any uniqueness that I might list for historical record. It isn't a technology, it isn't a science, and it isn't a marketing or business model. And I am going to give it dominant coverage in this paper.

As explained below, it is what I call my "Framework." It is based upon an intuitive conviction, emplanted in my head (apparently permanently) over thirty years ago, that the gains in human knowledge-work capability which we will achieve by properly harnessing this new technology will be very large. Metaphorically, I see the augmented organization or institution of the future as changing, not as an organism merely to be a bigger and faster snail, but to achieve such new levels of sensory capability, speed, power and coordination as to become a new species -- a cat.

Based upon this conviction about huge potential gains for mankind, my Framework explains for me generally where such gains are going to come from, and provides strategic principles that can help guide a conscious pursuit of these gains.

GENESIS

I was several years out of school, possessing a B.S. in EE and two years' experience during WWII (halfway through college) as an electronic technician. I was doing odd-job electrical engineering work at Ames Research Laboratory in Mountain View, California, with the National Advisory Committee for Aeronautics (NACA, forerunner of NASA). For several months I had been devoting most of my spare time to searching for professional goals; for some reason I wanted to invest the rest of my heretofore aimless career toward making the most difference in improving the lot of the human race.

I had initially dashed off in many fanciful directions, but yet managed enough interludes of

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1986 ACM-0-89791-176-8-1/86-0073 $00.75

reasonably sober thinking to build up some useful, strategic generalizations:

> Re-treading myself professionally, to become proficient and then extraordinarily productive in some new field wasn't worth considering without a significantly attractive scenario, embedded in a reasonably structured strategic framework.
>
> The high-payoff scenarios all seemed to involve creating or joining something that, however disguised, would essentially be a crusade.
>
> Crusades have many strikes against them at the outset. E.g., they don't connect to a normal source of government or business revenue; they don't have nice organizational frameworks -- you can't go out on the streets and expect to find financial, production, or marketing vice presidents; even if you accomplished the sweeping change that was the ultimate objective, chances are that in this very complex world, the side effects might be bad enough to make you wish you hadn't; etc.

Suddenly, up through all of this delightful, youthful abstraction bobbed the following clear realization:

> The complexity of the human situation was steadily increasing. Not only that, but its rate of increase was increasing.
>
> Along with the increasing complexity had come a general increase in the urgency associated with the more critical problems.
>
> If one invented a measure for each of these -- complexity and urgency -- then for a given problem, the product of its complexity measure times its urgency measure would represent a fair measure of the difficulty mankind would find in dealing with that problem.
>
> *FLASH-1: The difficulty of mankind's problems was increasing at a greater rate than our ability to cope. (We are in trouble.)*
>
> *FLASH-2: Boosting mankind's ability to deal with complex, urgent problems would be an attractive candidate as an arena in which a young person might try to "make the most difference."*
>
> Yes, but there's that question of what does the young electrical engineer do about it? Retread for role as educator, research psychologist, legislator, ... ? Is there any handle there that an electrical engineer could ...
>
> *FLASH-3: Ahah -- graphic vision surges forth of me sitting at a large CRT console, working in ways that are rapidly evolving in front of my eyes (beginning from memories of the radar-screen consoles I used to service).*

Well, the imagery of FLASH-3 evolved within a few days to include mixed text and graphic portrayals on the CRT, and on to extensions of the symbology and methodology that we humans could employ to do our heavy thinking; and also, images of other people at consoles attached to the same computer complex, simultaneously working in a collaboration mode that would be much closer and more effective than we had ever been able to accomplish.

Within weeks I had committed my career to *"augmenting the human intellect."* In a few months, I left the NACA and enrolled as a graduate student at UC Berkeley, where Professor Paul Morton had started a computer science activity (although it would be many years before universities began calling it that), and was several years along in developing the CALDIC.

Within a few years I had to accept the fact that research on any kind of interactive computer applications just wouldn't provide me with a program acceptable to the university community for PhD and later faculty pursuit. So, I settled for something else, got my PhD and went to Stanford Research Institute (SRI) where I hoped ultimately to promote support for an augmentation program.

FRAMEWORK

That was 1957. By 1959 I was lucky enough to get a small grant from the Air Force Office of Scientific Research (AFOSR, from Harold Wooster and Rowena Swanson) which carried me for several years -- not enough for my full-time work, but by 1960 SRI began pitching in the difference.

It was a remarkably slow and sweaty work: I first tried to find close relevance within established disciplines. For a while I thought that the emergent AI field might provide me with an overlap of mutual interest. But in each case I found that the people I would talk with would immediately translate my admittedly strange (for the times) statements of purpose and possibility into their own discipline's framework -- and when re-phrased and discussed from those other perceptions, the "augmentation" pictures were remarkably pallid and limited compared to the images that were driving me.

> For example, I gave a paper in 1960 at the annual meeting of the American Documentation Institute, outlining the probable effects of future personal-support use of computers, and how this would change the role of their future systems and also provide valuable possibilities for a more effective role for the documentation and information specialists <Pub-60-SpecCons>.
>
> No response at all at the meeting; one reviewer gave a very ho-hum description as ... the discussion of a (yet another) personal retrieval system. Later, at lunch during a visit to a high-caliber research outfit, an information-retrieval researcher got very hot under the collar because I wouldn't accept his perception that all that the personal-use augmentation support I was projecting amounted to, pure and simple, was a matter of information retrieval -- and why didn't I just join their forefront problem pursuits and stop setting myself apart.

Then I discovered a great little RAND report written by Kennedy and Putt <Ref-A> which described my situation marvelously and recommended a solution. Their thesis was that when launching a project of inter- or new-discipline nature, the researcher would encounter consistent problems in approaching people in established disciplines -- they wouldn't perceive your formulations and goals as relevant, they would become disputative on the apparent basis that your positions were contrary to "accepted" knowledge or methods, etc.

The trouble, said these authors, was that each established discipline has its own "conceptual framework." The enculturation of young professionals with their discipline's framework begins in their first year of professional school. Without such a framework, tailored for the goals, values and general environment of its respective discipline, there could be no effective, collaborative work. Furthermore, if such a conceptual framework did not already exist for a new type of research, then before effective research should be attempted, an appropriate, unique framework needs to be created. They called this framework-creation process the "Search Phase".

So, I realized that I had to develop an appropriate conceptual framework for the augmentation pursuit that I was hooked on. That search phase was not only very sweaty, but very lonely. In 1962, I published an SRI Report entitled "Augmenting Human Intellect: A Conceptual Framework" <Rpt-62J>. With the considerable help of Rowena Swanson, this was condensed into a chapter of a book published in 1963 <Pub-63-Frame>.

I can appreciate that these framework documents appear to many others as unuseably general and vague, but for me the concepts, principles and strategies embodied in that framework look better and better every year.

The genesis of most of what was/is unique about the products of the augmentation work can be traced back to this framework.

PROGRAM SUPPORT

I submitted many proposals before getting support to pursue the augmentation program outlined in the framework report. Among the stream of politely phrased regrets, there was one which in contrast to today's environment can provide useful perspective on the environment of 1961.

> Four high-quality civilian experts had been enlisted by one agency as a site-visit team; brain researcher, psychologist, computer expert -- and for me it was a very enjoyable day's dialog. But the later letter from the agency informed me regretfully that [paraphrased] *"... since your interesting research would require exceptionally advanced programming support, and since your Palo Alto area is so far from the centers of computer expertise, we don't think that you could staff your project adequately ...".*

When J. C. R. Licklider came from Cambridge to take over ARPA's newly formed Information Processing Techniques Office in late 1962, I was figuratively standing at the door with the Conceptual Framework report and a proposal. There the unlucky fellow was, having advertised that "man computer symbiosis," computer time-sharing, man-computer interface [illegible] were the new directions -- how could he in reasonable consistency turn this down, even if it was way out there in Menlo Park?

Lick moved very swiftly; by early '63 we had a project. But whereas I had proposed using a local computer and building an interactive workstation, Lick asked us instead to connect a display to the System Development Corporation's (SDC's) AN/FSQ32 computer, on site in Santa Monica, to do our experimenting under the Q32's projected new time-sharing system. (Converting the Q32 to be a time-shared machine was SDC's IPTO project.)

Later that year, our project was modified to include an online data link from Menlo Park to Santa Monica, with a CDC 160A mini-computer at our end for a communication manager, supporting our small-display workstation. For various reasons, not uncommon in pioneering ventures, that first year was very unproductive relative to the purposes and plan of our project. Lick was willing to put some more support into the direct goal (more or less as originally proposed), but the support level he could offer wasn't enough to pay for both a small research staff and some interactive computer support.

Mind you, the CDC 160A, which was the only commercially suitable mini-computer that we knew of, even though having only 8K of 12-bit words, and running at about 6 microseconds per instruction, cost well over $100K (1963 dollars). Paper tape in and out; if the system crashed, you had to load the application program from paper tape, and the most recent dump of your working file (paper tape), before you could continue. A crude, industry-standard Flexowriter (online typewriter) could be driven; otherwise it was paper-tape in and out.

What saved my program from extinction then was arrival of an out-of-the-blue support offer from Bob Taylor who at that time was a psychologist working at NASA Headquarters (then in Washington, D.C.). I had visited him months before, leaving copies of the Framework report and our proposal, and I had been unaware that meanwhile he had been seeking funds and a contracting channel to provide some support. The combined ARPA and NASA support enabled us to equip ourselves and begin developing Version 1 of what evolved into the NLS and AUGMENT systems.

Paul Fuhrmeister, and later Eugene Gribble of NASA's Langley Research Center, had to stick out their necks as successive heads of Langley's large computational division to support the direction and supervise NASA's support for our program, which continued several years after Taylor left NASA to join ARPA's IPTO office.

Our ARPA support grew and was fostered by Lick's successors -- Ivan Sutherland, Bob Taylor and Larry

Roberts. Meanwhile, the Air Force's Rome Air Development Center, at Rome, New York, began to supply supporting funds. By 1967, it was recognized that the respective contributions from ARPA, NASA and RADC represented significant parts of a coordinated program, and the other agencies began funneling their funds through RADC, which served for many years both with monitoring and managing our contracts, and providing their own significant share of support funds. Duane Stone and John McNamara provided strong support and contract liaison from RADC.

NASA support ended by 1969, and ARPA and RADC provided significant support until 1977, although from 1974 the support became ever more for supporting applications and developments for other organizations for targets formulated by others (e.g., the National Software Works) -- and the continuing pursuit of augmentation along my strategic vector virtually stopped.

THE AUGMENTATION RESEARCH CENTER (ARC)

An historically important organizational cluster emerged at Stanford Research Institute in the sixties, peaked about 1974, and was scattered in 1977 -- with a small core carrying forth in a commercial (and then industrial) environment to the present.

It grew by ones and twos from 1963, as it collected "permanent" members from the SRI technical staff, and recruited new ones from the outside. By '69 I believe we were about 18 strong; and this grew steadily until by '76 we totalled about 45. In 1973 we made two explicit sub-groups, one headed by Dick Watson doing development of software (and some hardware), and one headed by Jim Norton handling operations and applications support.

SRI was organized by divisions, each containing a group of laboratories; the hierarchy being formed according to the associated disciplines. ARC grew to laboratory size and status, but it became something of a problem for SRI.

Other laboratories (at least in science and engineering) operated more or less as a "farmers' market," where small and changing clusters of researchers promoted and conducted research projects as a loose federation. The management structure, budgeting, accounting and financing for the Institute had evolved to support this kind of business.

But ARC was driven by a coherent, long-term pursuit. This involved the continuing evolution of an ever-larger and more sophisticated system of hardware and software. It also came to involve delivering solid support service to outside clients to provide meaningful environments for learning about the all-important co-evolution processes in human organizations (human system & tool system).

It didn't seem unreasonable to me to pursue this course: things similar and on a grander scale are common for other researchers. E.g., it is taken for granted for funding agencies to build and operate accelerators and observatories in support of research in nuclear physics and astronomy -- or to outfit ships and airplanes to support research expeditions. But, whatever my perception, there were some significant problems and stresses for which our over-all environment didn't have effective ways to cope.

In the particular dynamics involved here, there were probably seven relevant parties: me; the ARC staff; other SRI researchers; SRI management and administration; ARC's sponsors; ARC's utility-service clients; and other groups of researchers outside of SRI.

It would be an interesting historical study to try to understand the diversity of perception that must have existed among this set of players. What did the different parties perceive for the future of workstations, for the range of function and application that would come about, for the systems architectures and standards that must emerge, and for the impact on the organizations that learned how to harness these most successfully?

Even as a central party in what happened, I've not understood the dynamics. But I am pretty sure that disparities among the perceptions of all of the above parties had a major part in what to me was the "great collapse of SRI-ARC." Even if I had done everything right over the years (a laughable hypothesis), it is now fairly clear to me that it isn't the market's fault if someone fails in trying to sell it something that it isn't ready for. In other words, I can't blame those other groups. (Which of course makes for a personal problem, since during those times of black discouragement when one wants desperately to blame someone, there is only one candidate -- that guy at the head of the list).

In 1977, SRI judged it better to move our large-system development and external-service activities out from the research institute environment and into a suitable commercial environment. They advertised, entertained prospective bidders, made a selection, and negotiated a transfer of the business to TYMSHARE, Inc., of Cupertino, California. The system was renamed AUGMENT, and marketed as part of TYMSHARE's integrated Office Automation services. In 1984, McDonnell Douglas Corporation acquired Tymshare, and the small AUGMENT business is now operated as the Augmentation Systems Division of the Computer Systems Company within the MDC Information Systems Group.

A CHRONOLOGY OF EVENTS

1962

* launched a long-term R&D program with the following basic Framework Principles (see <Rpt-62J>):

  There are many inventions, skills, methods, working conventions, organizational modes, etc.

which have been integrated into the human's knowledge-work environment to "augment" his basic, inherited, biological capabilities. Let a working assemblage of all of these be called an Augmentation System; my Framework considers this whole system to be a valid object of study and improvement.

I broke the many parts of the Augmentation System into two main sub-systems: one contained all of the hardware, software and other artifacts -- the Tool System; and all the rest of it I called the Human System. Note that the Human System contains our natural languages, and the conceptualizations and formalisms of every discipline: an overwhelming network of invention. (Sometimes, in the early years, I called these the Service System and the User System).

The emergent information technologies promised such startling innovation in the Tool System that, according to my Framework, most of the Human System elements that are involved in our knowledge work are likely to be up for improvement or replacement. And the Framework further predicted very large improvements in human knowledge-work capability as a result -- after we have gone through a few generations of evolution.

The challenging strategic question for me was: how best to invest whatever resources are available in a manner that best serves the evolution toward truly high performance by our knowledge organizations?

One answer: begin consciously scouting for elements in our Human System that are candidates for being changed as a means of better harnessing the technology toward our human ends.

This led directly to such things as our structured-text, with its links and views, and also to the mouse and the keyset.

"Bootstrap" leverage was another answer. Look for the innovations that will boost not only our regular productive capability, but will add as much as possible to our capability to further improve our Augmentation System.

This kept me focused on documents -- which carry the knowledge, plans, arguments, etc. that are critical to helping us better climb the evolutionary hill. It also very directly pointed to the importance of developing improved support for collaboration among distributed workers; and from there to community support, etc.

The whole Augmented Knowledge Workshop concept emerged from the Bootstrap strategy -- working toward coherent, integrated access to an open-ended, evolving collection of resources and people. And doing this in a way that best enables evolutionary freedom of parts of the whole system without being unnecessarily anchored by the needs of other parts. From this arose such things as the Procedure Call Protocol, the application-independent User Interface System, and the Network Virtual Terminal, etc.

But read on ...

1963

* developed the chord keyset as a one-handed alternative for character input in an interactive environment. See <Rpt-65C> and <Pub-67-DispSel>.

* I participated in the ARPA-sponsored "summer study group" that kicked off Project MAC at MIT. Memories:

  Wrote a project memo about the dichotomy between "User System" and "Service System." Essentials are described in <Rpt-65C>, and the expanded dichotomy recurs as for instance in <Pub-73-AKW>, <Pub-78-IntEvolOAS>, and <Pub-82-HiPerfKW>.

  Had my turn to address the whole group (50 or so people); not much reaction, but I do remember the group hoot when I said that we'd all be seeking to boost response time for many man-computer interactions down to at least a quarter of a second before the returns began to diminish.

* tried unsuccessfully to develop a workable system using a long-distance data link from Menlo Park to Santa Monica, with the SDC Q32 as a time-sharing host, and a CDC 160A as local communication manager and display driver.

1964

* made explicit strategic decision to bypass the online typewriter and go directly to display workstations. In spite of most other online developers working with typewriters, I felt that the much higher "augmentation potential" of displays warranted early pursuit, and figured that cost would surely come down in a few years. Keep in mind that CRT workstations were very expensive then -- they weren't a consumer product by any means. The single, large-CRT workstation we attached to the CDC-3100 that year probably cost us over $100,000, and required a lot of custom work on our part. [But we have always kept an access mode open for "dumb terminals" and "glass teletypewriters," which can elicit as much service as possible within their limitations.]

* moved to stand-alone CDC 160A; paper-tape I/O to and from a Flexowriter paper-tape typewriter; first, primitive version of our online, structured-file editing. Had a simple, batch system, too: type in correction directions on a paper-tape-punching Flexowriter; feed that paper tape into the 160-A for pre-processing; results and the file to be corrected transferred via mag tape to SRI central processor (Burroughs 220, I think); processed results brought back for post-processing and printing on the 160A.

1965

* developed the mouse as part of an explicit search for optimum screen-selection techniques in association with our online-application framework. See <Pub-67-DispSel>

* moved NLS to a stand-alone CDC-3100, with online disk pack, 16K of 24-bit memory, line printer, paper-tape and punched-card I/O, custom built display. Full structured files, with in-file addressing and uniform text- and structure-manipulation commands.

* Published <Rpt-62J>, which included the following:

  Discussion of how the computer can give aid to the basic communication processes between the human and his external environment (which environment of course includes his kit of computer tools). Clear forerunner of our later User Interface System.

  Uses example of introducing a chord keyset into a user's environment to bring out specific examples of the NEED-POSSIBILITY REVERBERATION phenomena which was introduced in 1962, <Rpt-62J>, and which is a basic part of all my subsequent strategic thinking.

  Extensive discussion of USER-SYSTEM, SERVICE-SYSTEM DICHOTOMY, including the characteristics of research in each domain. (CO-EVOLUTION without yet naming it such.)

  Detailed description of the earliest version of our later FLTS (oFf-Line Text System), and what eventually became our DEX System (Deferred EXecution). Working at a Model-33 Teletype, which makes a punched-tape record of all that is typed. Escape and Command codes embedded anywhere in the text could cause the later "batch" process to correct any previous error or omission, including those in any prior commands. The Model-33 had only one case of alphabetic characters; this process enabled me to designate cases so that the later, processed printout on a two-case Flexowriter (paper-tape driven) would come out with proper alphabetic cases.

  (I made myself write most of that report this way. And my wife made me move out into the garage to do it because the Model-33 was so noisy. I remember the extra problem of typing with cold fingers.)

* Showed real time movie of NLS to ARPA IPTO Principal Investigators at our May 16-17 meeting at MIT. Simple, structure-text manipulations, with very fast concurrent control (mouse and keyset), and very fast computer response (stand-alone CDC-3100). Illustrated the basic difference in perspective between our approach and the prevailing concepts of "time-shared computer support." At that evening's cocktail hour, Bob Taylor told me, *"The trouble with you, Doug, is that you don't think big enough."*

  Well, after he dragged out of me a description of what I'd really like, he encouraged me to formulate a proposal for it -- a multi-user laboratory, based on a time-shared machine, to get on with real bootstrapping.

1966

* all source-code development, maintenance and documentation now fully moved into the NLS environment (providing natural and powerful support for structured programming).

* began planning for move to time shared environment;

1967

* at Spring meeting of ARPA Principal Investigators, in Ann Arbor, during session kicking off ARPA's newly revealed Network development project, I volunteered to develop and operate a Network Information Center (NIC) -- as a special, prototype support service for the network-connected community of ARPA-IPTO researchers.

* developed our first explicitly separated command processor (I used the term "control" then, instead of "command"), supported with our "Control Meta-Language," a separate command-description, compileable language. Cf. Branch 4c and Fig. 9 of <Pub-68-ResCen>.

* developed our MOL 940, a Machine Oriented Language and compiler for the SDS 940.

* equipped a meeting room with specially-built tables, video hookup, and computer controls for real-time support of meetings with our CDC-3100 version of NLS. (Photo and description in <Rpt-68D> and in <Ref-B>.)

  Over twenty people could participate, each had full view of a nearby video screen; each could pick up a mouse to control a specially shaped "participant's cursor" so all could see what he was pointing at; one master control station provided discussion-recorder control for accessing, flexible viewing, and modification of project documents, agendas, formulation of measures to act upon, etc.

  We used this support system on 12-13 October, 1967, for a progress-review meeting with our research sponsors: Bob Taylor and Barry Wessler from ARPA, Fred Dion and Dean Bergstrom from RADC, and Gene Gribble from NASA.

1968

* NLS by now had a full set of basic features that have since characterized it and AUGMENT (its commercial successor). E.g., full-screen, integrated outline and text processing; in-file addressing; in-file and cross-file text or structure manipulation by address; basic repertoire of view-control commands; content filtering; generalized, computer-executable citation links; verb-noun, consistent command syntax with optional use of ultra-fast,

concurrent control using the mouse and chord keyset;

Included also was a calculator package, integrated into NLS: mouse-selecting operands; totaling columns; inserting accumulator contents at selected locations or replacing selected numbers in a file; executing user macros with pauses and prompts for users to select file variables or provided typed in values.

* Put together our home-designed, custom-built displays system to run with the SDS-940.

Two custom-built, random-deflection display generators were each time-shared to drive six, small, 5" high-precision CRTs. In front of each of these CRTs was mounted a high-quality, video camera so as to scan the CRT face. These twelve video lines were brought out to our work area, where each work station had a high-quality video monitor for its display.

This gave us four sizes of alphanumeric characters, and accelerators vector-graphic figures. The display generators were connected on a Direct Memory Access bus so that switching from one stored view to another occurred essentially in a thirtieth of a second.

* came online with an improved NLS on a time-shared, SDS 940; large swapping drum; special, home-made display system operating from direct-memory access, providing integrated text and graphics, and delivering video to up to twelve workstations out in our laboratory.

* Made a public debut at the Fall Joint Computer Conference in San Francisco, December. For this event, we added another layer of new technology on top of NLS, a system that was already very complex for its day. It is worth an extra bit of description here.

Bill English and I wrote a paper for this conference describing ARC's objectives, physical laboratory, and the current features of NLS. In the Spring, when the Program Committee was considering candidate papers and organizing its sessions, I also proposed that they let us have a full hour-and-a-half session to put on a video-projected, real-time presentation. After considerable deliberation, and no less than two site visits to our lab at SRI, they consented.

It was a considerable gamble, possibly an outright misuse of research funding. I have no illusions that it could possibly have been pulled off without Bill English's genius for getting things to work. Our new display system provided us with twelve video cameras; we left about half of them working as display generators, and used the others to provide video views of people, borrowing tripods and drafting all kinds of people as camera operators and prompters.

We leased two video links to send images from SRI to the Conference Center in San Francisco -- a direct distance of about 30 miles. It required temporarily mounting four pairs of dishes -- two atop our SRI building, two atop the Conference hall, and four on a truck parked on top of a relay mountain. We procurred some video-lab equipment: frame splitters, switches, faders, and mixers. We made special electronics to get our mouse and other terminal signals from the podium to the 940 at SRI.

It required a special video projector, whose rental included a specialist from New York to set it up and operate it. He proved invaluable in making other things work that day, too. Two cameras were mounted on the stage where I sat at the special work station (which the Herman Miller Company had made for us, and donated).

I was on-stage as anchor man during the continuous, 90-minute presentation, and Bill sat in the canvas-enclosed, raised booth at the back of the auditorium, directing the participants according the the script that I had prepared. People in our laboratory had key roles, and Bill coordinated us all via a voice intercom, while he also did the switching and mixing and frame splitting to put together the projected images.

During that 90 minutes, we used the projected display images (composite text & graphics) both to present agendas and descriptive portrayals, and also to demonstrate what NLS could do and how we applied it to our planning, documenting, source-code development, business management, and document retrieval.

The audio and video of the entire presentation was captured on film (no portable video recorders in those days). We had ten prints made, and circulated free loaners to people for years. (This film has recently been converted to VHS video cassette form to facilitate viewing by other people.)

1969

* began design of windowing capability for NLS.

* developed concept of a user "reaching through" his personal workplace (i.e., his familiar online working files and application programs) to access less basic, specialized data and application processes (and other people); i.e., the "reach through" should provide access to these, translated by the integrated support system, so as to appear as coherent parts of his familiar, personal workplace.

* specified our first mail and "Journal" system as part of an explicit pursuit of a "Dialog Support System," planning for it to be part of our ARPANET-NIC service.

* Developed document-outputting capability processing our composite, text-graphic document files to drive a service-bureau, CRT-based, full-page, Stromberg-Carlson photo printer to produce documentation with graphics and text mixed on the same pages.

* Became the second host on the ARPANET with our SDS 940. (UCLA was first, UCSB next, then the University of Utah, then ....)

1970

* (See also <Rpt-70D> and <Rpt-72F>.)

  Detailed use of NLS for internal management processes of ARC: Cost records, working forecasts, purchase Renaissance, etc.

  In 1970 we began using the ARPANET to facilitate our re-programming of NLS for the forthcoming PDP-10 TENEX. The University of Utah had a TENEX on the network, and we used NLS on the 940 to write our new PDP-10 code; using our Tree-Meta compiler, we developed a cross-compiler for our 940 that produced PDP-10 relocatable binary code. We would ship that over the net for loading and debugging on Utah's TENEX.

  When the two computers and the intervening network link were all working properly (lots of flat tires in the early days of automobiles), our programmers would do all of this back and forth transitioning "through" the same workstation.

  I think that it was not only a record-making way of working, but the NLS transport task was accomplished in remarkably short time (we attributed part of the efficiency to the network, and part to the use of NLS).

  In late 1970 we brought NLS up on the PDP-10 TENEX with improved and new features (including multiple windows).

  The transfer process, and a detailed description of the design changes and new features for NLS are described in <Rpt-71F>.

* began using our Mail/Journal system within our group. Integrated into NLS, this assumed that a mail item was a document -- so any part of all of an NLS document could be sent. Provided for permanent record in explicitly retrievable form (our Journal). As an electronic-mail system, this was quite advanced. It had a Directory service (our Ident System) to provide mail-relevant information about registered users; mail distribution was addressed by people's Idents, with no need to know or specify which host they used. Fields were provided for superceding other items, and for attaching keywords. An online index was provided for stored items. See descriptions in <Rpt-71F> and in <Pub-75-TelConf>.

1972

* began developing our first, integrated Help system.

* formulated the "AKW Architecture" -- implemented in stages, described in <Pub-82-HiPerfKW>.

* implemented the "shared-screen," televiewing mode of online collaboration between two or more NLS users. See <Pub-68-ResCen>, <Pub-75-TelConf> and <Pub-78-IntEvolOAS>.

1973

* brought up a Table Subsystem in NLS.

* designed our first, totally modular User Interface System, as later described in <Pub-82-HiPerfKW>. Got it running on a PDP-11 that talked to our TENEX through the network, via our Procedure Call Protocol.

* Developed our Line Processor, as described by Don Andrews in <Pub-67-DispSel>. It incorporated Intel's first microprocessor (the 4004) in a special box which was inserted in the communication line between a dumb display terminal and a modem. This made use of our Virtual Terminal Protocols, and managed a multi-window, two-dimensional screen using off-the-shelf, "dumb" display terminals. Our mouse and keyset input devices were plugged into the line processor, which appropriately translated their actions to control cursor position and special communications to the host. A printer port on the line-processor provided local printout service; a special communication protocol allowed the host to send printer packets mixed in with display-support packets.

* Finalized specification for our Network Virtual Terminal, something which has become a key part of our architecture. The objective on the one hand was to free the application programmers from worrying about the special features of different workstations, and on the other hand, to enable more flexible evolution by users of workstations they may adopt to fit particular needs. As part of this, there was a terminal-independent Display Manipulation Protocol for communication from application program to terminal, and an application independent Input Protocol for communicating from terminal to application program.

* generalized the file structure of our document files to provide for generalized property structures associated with each addressable object; intended to accommodate composite integration of such as graphics, digitized speech, scan-coded images, or any other arbitrary data form.

1974

* gave up our high-performance, local display system for the line-processor supported remote display system -- to make ourselves live with the same remote services as our NIC clients and Utility customers. [On principle, we gave up our integrated, direct-view graphics and the fast response of our direct-memory-access, local display generator.]

* opened our "Workshop Utility Service" -- delivering NLS service over the ARPANET to DoD customers as pilot applications of office

information service; had gone out on bid for commercial time-sharing services, selected Tymshare Inc. of Cupertino, Ca.; their host, named Office-1 provided the computer service; we fielded special trainers and application development staffs and cultivated special customer representatives into a spirited community.

1975

* implemented our new, integrated graphics system, which could support remote display and manipulation of illustrative graphics on a Tektronix 4014 storage-tube display plugged into the line-processor's printer port. The graphic data were embedded within the NLS structured-text files; an illustration produced as a plotter-driver file by any other graphics system could be picked up and attached to a specified location in an NLS document, and be subsequently viewed and modified. Also, our Output Processor could direct that a properly scaled image of a each such illustration be located appropriately within a multi-font page layout.

CONCLUSIONS

I hadn't realized just how much history there is back there in my past. I wish that I could have done a more uniform job of digging and organizing for this paper. Oddly enough, a major problem is that I am quite distracted by the needs and opportunities for working on tomorrow's history -- there is still such a large "unfinished agenda" stemming from my compulsive commitment to the Augmentation Framework.

Re-living the past during this history exercise has left me with a feeling of having re-visited the Dark Ages. Paper tape, expensive technology, and years of frustrations with flaky, first-of-a-kind systems -- thank God that we have now emerged into the Rennaisance.

But another side of my experience has been with what people perceive about the possibilities stemming from these new technologies. And I am not so sure that we have moved all that far out of the Dark Ages in that regard.

I still don't see clear perceptions about what we humans can gain in new capabilities, or about how this may come about. There are constant, echoing statements about how fast and smart the computers are going to be, but not about how the enhanced computer capabilities will be harnessed into the daily thinking and working life of our creative knowledge workers.

I guess what I am hoping to see is the emergence of professional societies concerned with something like what I call a whole Augmentation System. And I'd like there to be special conference sessions and research efforts focused on pursuit of very high performance for teams of people equipped with integrated new technologies AND with new concepts, methods, roles, skills, etc.

And I'd like to hear people begin talking about the new OPPORTUNITIES to change their ways and skills so that they can more EFFECTIVELY HARNESS the new tools, in pursuit of their own INCREASED EFFECTIVENESS. I very much look forward to harnessing smarter and smarter computer processes within my working domain, but I would like to hear less about making computers very smart so that (apparently) the user won't have to learn new things.

PARTIAL CAST OF CHARACTERS

We recently had a reunion of all the people that had worked with NLS/AUGMENT over the years. We found about 150 names when we dug back in records and people's memories. Many of those people gave spirited support and since have gone on to work at other kinds of jobs. And many have stayed in a similar job area and have continued to contribute elsewhere -- publishing, developing, entrepreneuring and consulting.

Here is an alphabetic listing of some of these latter, stemming from the more historical era:

Don Andrews
Jim Bair
Roger Bates
Bob Belleville
Dave Casseres
Jan Cornish
Tom Davis
Chuck Dornbush
Bill Duvall
Joe Ehardt
Bill English
Jake Feinler
Larry Garlick
Martin Hardy
Ken Harrenstein
Dave Hopper
Charles Irby
Susan Kahn
Kirk Kelley
Harvey Lehtman
Rob Lieberman
Dave Maynard
John Melvin
Dean Meyer
Elizabeth Michael
Ray Panko
Bruce Parsely
Bill Paxton
Jeff Peters
Andy Poggio
Jon Postel
Raphi Rom
Jeff Rulifson
Glenn Sherwood
Dave Smith
Jacques Vallee
Dirk Van Nouhuys
Ken Victor
Smokey Wallace
Dick Watson
Anne Weinberg (Duvall)
Jim White

REFERENCES

NOTE: Arrangements are being made to collect all of the records and films from the SRI-ARC era and store them in the History-of-Science archives at the Stanford library. This will include all of the Journal items generated by ARC staff, which is a very unique collection of detailed history beginning in 1970. These archived records will be available to qualified researchers; access will be controlled by the library archivist.

RPT: Some Relevant SRI-ARC Reports

<Rpt-62J>: "Augmenting Human Intellect: A Conceptual Framework," Summary Report, Stanford Research Institute, on Contract AF 49(638)-1024, October 1962, 134 pp, Engelbart, D. C.

<Rpt-65C>: "Augmenting Human Intellect: Experiments, Concepts, and Possibilities," D. C. Engelbart; Summary Report, Stanford Research Institute, under Contract AF 49(638)-1024 for Directorate of Informtion Sciences, Air Force Office of Scientific Research, March 1965. SRI Project 3578; 65 p. <XDOC, 9691,>

<Rpt-65G>: "Computer-Aided Display Control," W. K. English, D. C. Engelbart, Bonnie Huddart, SRI Final Report to NASA Langley Research Center under Contract NAS 1-3988, July 1965, 104 p. <XDOC, 9692,>

<Rpt-68D>: "Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research," D. C. Engelbart, W.K. English, J.F. Rulifson, SRI Final Report for AF Rome AIr Development Center under Contract AF 30(602)-4103, April 1968, nnn p., <XDOC, 9697,>

<Rpt-70D>: "Computer-Augmented Management-System Research and Development of Augmentation Facility," D. C. Engelbart, Final Report, RADC, <AUGMENT, 5139,>

<Rpt-71F>: "Network Information Center and Computer Augmented Team Interaction," Technical Report RADC-TR-71-175, by D. C. Engelbart as principal investigator for SRI Contract F30602-70-C-0219 under ARPA Order No. 967. June 1971, 99p. <AUGMENT, 8277,>

<Rpt-72F>: "Online Team Environment: Network Information Center and Computer Augmented Team Interaction," D. C. Engelbart, Final Report RADC-TR-72-232 to Air Force Rome Air Development Center for SRI work under ARPA Order No. 967., June 8, 1972, 264 p. <AUGMENT, 13041,>

PUB: Open-Literature, "Augmentation" Publications

<Pub-60-SpecCons>: "Special Considerations of the Individual as a User, Generator, and Retriever of Information," American Documentation, 12, No. 2, pp. 121-125, April 1961. [Paper presented at the Annual Meeting of the American Documentation Institute, Berkeley, Calfornaia, October 23-27, 1960.], Engelbart, D. C.

<Pub-63-Frame>: "A Conceptual Framework for the Augmentation of Man's Intellect," in Vistas in Information Handling, Howerton and Weeks (Editors), Spartan Books, Washington, D. C., 1963, pp. 1-29. Engelbart, D. C.

<Pub-67-DispSel>: "Display-Selection Techniques for Text Manipulation," IEEE Transactions on Human Factors in Electronics, Vol. HFE-8, No. 1, March 1967, Pp. 5-15. English, William K., Douglas C. Engelbart and Melvyn L. Berman.

<Pub-68-ResCen>: "A Research Center for Augmenting Human Intellect," AFIPS Conference Proceedings, Vol. 33, Fall Joint Computer Conference, San Francisco, December 1968, pp. 395-410 FJCC 1968. <AUGMENT,3954,>, Engelbart, D. C., W. K. English

<Pub-70-IntImpl>: "Intellectual Implications of Multi-Access Computer Networks," Proceedings of the Interdisciplinary Conference on Multi-Access Computer Networks, Austin, Texas, April 1970 <AUGMENT,5255,>, Engelbart, D. C.

<Pub-72-CommServ>: "Coordinated Information Services for a Discipline- or Mission-Oriented Community," Proceedings of the Second Annual Computer Communications Conference in San Jose, California, Jan 24, 1972 -- also, Published in "Computer Communication Networks," R. L. Grimsdale and F.F. Kuo editors, Noordhoff, Leyden, 1975. <AUGMENT,12445,> Engelbart, D. C.,

<Pub-73-AKW>: "The Augmented Knowledge Workshop," AFIPS Conference Proceedings, Volume 42, pp. 9-21, National Computer Conference, June 4-8, 1973. Engelbart, D. C., R. W. Watson and J. C. Norton. <AUGMENT,14724,>

<Pub-73-KWTermDes>: "Design Considerations for Knowledge Workshop Terminals," AFIPS Conference Proceedings, Volume 42, pp. 221-227 , National Computer Conference, June 4-8, 1973. Engelbart, D. C., <AUGMENT,14851,>

<Pub-74-LineP>: "Line Processor -- A device for amplification of display terminal capabilities for text manipulation," AFIPS Conference Proceedings, pp. 257-265 , National Computer Conference, 1974. <AUGMENT,20184,> Andrews, Donald I.,

<Pub-75-TelConf>: "NLS Teleconferencing Features: The Journal, and Shared-Screen Telephoning," Engelbart, D. C., CompCon75, Sept. 9-11 '75, Digest of papers, pp.173-176. [IEEE Catalog No. 75CH0988-6C] <AUGMENT,33076,>

<Pub-76-HLFrame>: "A High-Level Framework for Network-Based Resource Sharing," James E. White, J. E., AFIPS Conference Proceedings, June 1976, NCC, Volume 45, pp. 561-570. <AUGMENT,34263,>

<Pub-76-UIDes>: "User Interface Design Issues for a Large Interactive System," Richard W. Watson, AFIPS Conference Proceedings, Volume 45, AFIPS Press, 1976, Montvale, NJ, pp. 357-364. <AUGMENT,27171,>

<Pub-77-DAD>: "The Design and Implementation of DAD, A Multiprocess, Multimachine, Multilanguage Interactive Debugger," Victor, K. E., Proceedings of the Tenth Hawaii International Conference on System Sciences, University of Hawaii, 1977, pp. 196-199. <AUGMENT,29079,>

<Pub-77-SEngrEnv>: "A Software Engineering Environment," Kenneth E. Victor, Proceedings of AIAA/NASA/IEEE/ACM Computers In Aerospace Conference, Los Angeles, CA, October 31-November 2, 1977, pp. 399-403. <AUGMENT,29292,>

<Pub-78-IntEvolOAS>: "Toward Integrated, Evolutionary Office Automation Systems," Proceedings of the Joint Engineering Management Conference, Denver, Colo., 16-18 Oct 78, pp 63-68. <AUGMENT,71279,>, Engelbart, D. C.

<Pub-80-EvolOrg>: "Evolving the Organization of the Future: A Point of View," A paper published in "Emerging Office Systems," Proceedings of the Stanford International Symposium on Office Automation, March 23-25, 1980, edited by Robert Landau, James Bair, Jeannie Siegman, Ablex Publications Corporation, Norwood, NJ. <AUGMENT,80360,>, Engelbart, D. C.

<Pub-82-HiPerfKW>: "Toward High-Performance Knowledge Workers," Douglas C. Engelbart, OAC'82 Digest, Proceedings of the AFIPS Office Automation Conference, San Francisco, April 5-7, 1982, pp. 279-290. <AUGMENT,81010,>

<Pub-84-AuthSupp>: "Authorship Provisions in AUGMENT," Douglas C.Engelbart, COMPCON '84 Digest, Proceedings of the 1984, COMPCON Conference, San Francisco, Ca., February 27 - March 1, Pp. 465-472. <OAD,2250,>

<Pub-84-CollabSupp>: "Collaboration Support Provisions in AUGMENT," Douglas C. Engelbart, OAC '84 Digest,Proceedings of the 1984, AFIPS Office Automation Conference, Los Angeles, Ca., February 20-22, Pp. 51-58. <OAD,2221,>

REF: Other-Author Publications

<Ref-A>: "Administration of Research in a Research Corporation," J. L. Kennedy and G. H. Putt; RAND Corporation Report P-847 (20 April 1956)

<Ref-B>: "The Computer as a Communication Device," from J.C.R. Licklider and Robert W. Taylor, with Evan Herbert ed., Science & Technology, April 1968, pp. 21-31.

<Ref-C>: "As We May Think," Vannevar Bush; The Atlantic Monthly (July 1945)

<Ref-D>: "Man-Computer Symbiosis," J. C. R. Licklider; IRE Transactions on Human Factors in Electronics (March 1960).

<Ref-E>: "On-Line Man-Computer Communication," J. C. R. Licklider and W. E. Clark; Proceedings Spring Joint Computer Conference, Vol. 21, pp. 113-128 (National Press, Palo Alto, California, May 1962).

<Ref-F>: "Solution on Non-Linear Integral Equations Using On-Line Computer Control," G. J. Culler and R. W. Huff; paper prepared for Spring Joint Computer Conference, San Francisco, May 1962. [Assumedly published in Proceedings Spring Joint Computer Conference, Vol. 21, (National Press, Palo Alto, California, May 1962).]

<Ref-G>: "Augmenting Your Intellect" -- Editorial interview with Douglas C. Engelbart, in Research/Development (Magazine), August 1968, pp. 22-27.

<Ref-H>: "Toward the Decentralized Intellectual Research/Development (Magazine), Article by Nilo Lindgren, staff writer, in Innovation, No. 24, September 1971. (XDOC, 10430,)

<Ref-I>: "The Origin, Anatomy, and Varieties of Mus Computeralis," (It's the Year of the Mouse!), by Tom Foth, Softalk, Vol. 2, April 1984, Pp.88-96.

<Ref-J>: "Of mice and Men," Steven Levy, Popular Computing magazine, May 1984, pp. 70, 75-78.

<Ref-K>: "The Loneliness of a Long-Distance Thinker," Chapter Nine, pp. 174-204, in book "Tools for Thought: The People and Ideas behind the Next Computer Revolution," by Howard Rheingold, Simon & Schuster, New York, 1985, 335 pp.



The Dynabook - Past, Present, and Future

Alan Kay

Apple Computer, Inc.

Paper not received in time for inclusion in the Conference Proceedings.
A full paper will be distributed at the Conference.

Personal Distributed Computing: The Alto and Ethernet Hardware

Charles P. Thacker
Systems Research Center
Digital Equipment Corp.
130 Lytton Avenue
Palo Alto, CA 94301

Abstract

Between 1972 and 1980, the first distributed personal computing system was built at the Xerox Palo Alto Research Center. The system was composed of a number of Alto workstations connected by an Ethernet local network. It also included servers that provided centralized facilities. This paper describes the development of the hardware that was the basis of the system.

Introduction

In the last few years, a new type of computing environment has become available. These *distributed personal computing* systems represent another step in the process, started by timesharing, of bringing computing power closer to the user. Although many variations are possible, these systems share a number of characteristics:

- They are based on *workstations*—personal machines that are sufficiently powerful to satisfy essentially all the computational needs of a single user. The workstations include high resolution displays, and provide a highly interactive user interface.
- The workstations are interconnected by *local networks* that provide high bandwidth communication throughout a limited area, typically a single building.
- In addition to the workstations, the network contains *servers*—nodes that provide capabilities that need to be shared, either for economic or logical reasons.

Timesharing systems grew primarily from the vision of man-computer symbiosis presented by J.C.R. Licklider in a landmark 1960 paper [29]. Efforts to realize the possibilities presented in this paper occupied the creative talents of many computer science researchers through the sixties and beyond. Distributed personal computing systems build on this view of the way computers and people interact by providing a level of responsiveness that timesharing systems cannot achieve.

The first distributed personal computing system was built at the Xerox Palo Alto Research Center over a period spanning 1972 to 1980. The workstation used in this system was the Alto [36]; the network was Ethernet. This paper describes the hardware that was the foundation of this system. A companion paper by Butler Lampson [24] describes the software that was built on the hardware base described here.

This paper contains seven sections: Section 1 describes the environment in which the work was done. Section 2 traces some of the underlying ideas. Section 3 describes the early implementation period, and section 4 discusses the servers that provide printing and file storage in the system. In section 5, the reengineering effort that made the Alto into a successful internal product is described. Section 6 briefly discusses some of the Alto's successors, and section 7 contains concluding remarks.

1 The Environment

The Xerox Palo Alto Research Center (PARC) was established in 1970, primarily through the efforts of Jacob Goldman, director of corporate research. It was composed of three laboratories: the Computer

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0087 $00.75

Science Laboratory, the System Sciences Laboratory, and the General Science Laboratory. To direct the new center, Goldman recruited George Pake, a physicist who was at that time provost of Washington University, St. Louis.

To establish the Computer Science Lab, Pake engaged Robert Taylor, who had directed the Information Processing Techniques Office of ARPA during the late sixties. Taylor had worked with and funded many of the leading computer science research groups during this period. As a result, he was in a unique position to attract a staff of the highest quality.

During the first year of CSL, Taylor built a group of approximately fifteen researchers, drawn from MIT, the University of Utah, and CMU. Several members of CSL came from Berkeley Computer Corporation, a start-up company composed primarily of individuals from the University of California at Berkeley, who had built one of the first timesharing systems [25]. From Bolt, Beranek and Newman, Taylor later brought Jerome Elkind, who was the manager of CSL from 1971 until 1976. Also during this period, Alan Kay, who was to provide much of the vision on which the Alto was based, was recruited into the Systems Sciences Laboratory by Taylor. Kay established the Learning Research Group (LRG), and defined its goal: To produce a programming system in which "...simple things would be simple, and complex things would be possible [20]."

The research environment built by Taylor was one of the main reasons for the success of CSL and its projects over the next decade. Unlike other PARC laboratories, CSL was not organized into permanent groups. Instead, researchers were encouraged to move between projects as their talents and the needs of the projects dictated. This flat structure and the mobility it made possible encouraged members of the lab to become familiar with all activities. Additionally, it provided a continuous form of peer review. Projects which were exciting and challenging obtained something much more important than financial or administrative support; they received help and participation from other CSL researchers. As a result, quality work flourished, less interesting work tended to wither.

During 1971 and early 1972, most of the effort in CSL was spent in building a set of hardware and software facilities to support the future work of the laboratory. The Maxc timesharing system [13] was built and the Tenex [2] operating system was acquired and modified for it. Projects in graphics, computer networking, and language design were started.

The main research theme of CSL—office information systems—was also developed during this period. Most of the research done in CSL and SSL during the next five years was organized around this theme, which reflected the desire of Xerox to expand its traditional copier business to include most of the functions performed in offices. Eventually, this theme was broadened to include what is now known as distributed personal computing, but initially our ambitions were lower.

A strategy for carrying out work in experimental computer science was also adopted at this time. It was based on the idea that demonstrations of "toy" systems are insufficient to determine the worth of a system design. Instead, it is necessary to build *real* systems, and to use them in daily work to assess the validity of the underlying ideas, and to understand the consequences of those ideas. When the designers and implementors are themselves the users, as was the case at PARC, and when the system is of general utility, such as an electronic mail system or a text editor, there is a powerful bootstrapping effect. This view of systems research is quite different from that found in most academic environments, since it requires larger groups working over a longer period than a university can usually support.

It was clear that the ability to provide systems with high levels of functionality would be limited by software considerations far more than by the capabilities of the underlying hardware. While it would be necessary for us to build hardware, since the needed capabilities were not commercially available, the characteristics of the devices would be determined primarily by the needs of the software systems for which they were intended. This view is commonplace today, as hardware performance has increased and its cost has declined, while the cost of delivering large, reliable software systems has continued to increase. It was much more radical in 1971.

Although hardware development was an integral part of the work carried out in CSL, it represented a small fraction of the overall activity of the laboratory. At no time did the number of people engaged primarily in hardware work exceed five—roughly ten percent of the total professional staff. This core group was very effectively augmented during large projects (e.g., the development of the Dorado [27] during 1977-80) by laboratory members with computer science, rather than electrical engineering backgrounds. However, most things were done by a relatively small group. For this reason, it was necessary to be selective in our choice of projects. Simplicity and utility were the most important criteria. Highly complex designs would have been beyond our capabilities, and the construction of systems without a wide user community would not have justified the expenditure of scarce design and implementation talent.

2 Sources of Ideas

By late 1972, most of the laboratory facilities were in place in CSL, and the researchers who had produced them began planning longer-term projects. It was at this time that the main ideas underlying the Alto were discussed and refined into an actual design. Although a number of people in CSL and SSL contributed to the specification of the new system, Butler Lampson, Alan Kay, and Bob Taylor were the individuals who were primarily responsible for shaping the design. To the extent that CSL had project managers, I filled that function. My task was to convert the vision of Lampson, Kay, and Taylor into working hardware.

Taylor had originally proposed that the primary computing facility in CSL should be an interconnected collection of small display-based machines, rather than a central timesharing system. He thought that a sufficient amount had been learned about the design of interactive systems, and that hardware costs were low enough that it would be feasible to produce such a system in modest quantities. In 1970, this idea contained a number of technical difficulties. Lampson and I argued that a stable set of computing facilities as well as experience in producing hardware in the new laboratory would be required before embarking on such an ambitious project. This conservative view prevailed, but by 1972, sufficient progress had been made in a number of areas, particularly semiconductors, that the difficulties did not seem as overwhelming.

In his 1969 thesis, Alan Kay had described a small computer system, the "reactive engine" [21], that shared many of the characteristics of the new machine. Like Taylor, Kay wanted a system that would provide a complete work environment for its user, including text and graphics manipulation, computing, and communications capability. By 1972, this vision of computing had acquired a name—Dynabook—and for a while, the Alto was called the "interim Dynabook" by its developers. Kay's vision was that the ultimate Dynabook would be portable, so that it could provide all the functions provided by books, paper, pencil, and terminals. Although the Alto never achieved this part of his vision, it served for a number of years as the hardware environment for the Smalltalk system [16], in which a number of text and graphics, music, and simulation applications were built.

Lampson's view of the capabilities of the new system and its uses was perhaps the most explicit. In a 1972 guest editorial [23] in Software–Practice and Experience, he had predicted that within five years, it would be possible to build a system "...comparable to a 360/65 in computing power for a manufacturing cost of perhaps $500." In the same article, he predicted significant advances in programming technology, and foresaw the some of the effects that would follow from these developments:

> As a result, millions of people will write nontrivial programs, and hundreds of thousands will try to sell them... Almost everyone who uses a pencil will use a computer, and although most people will not do any serious programming, almost everyone will be a potential customer for serious programs of some kind.

Although it was clear that we could not achieve these goals in 1972, there was a clear consensus that the new system should have characteristics that were prototypical of this vision. Cost was not a primary consideration in the design, but it could not be outrageous, since the system had to be producible in modest quantities to justify the development of software for it.

By late 1972, the principal features of the new machine had been defined. The major departure from past systems was the machine's display (see Figure 1). To emulate as many of the characteristics of paper as possible, we chose to provide a full bitmap, in which each screen pixel was represented by a bit of main storage, and to use raster scanning rather than the lower-cost calligraphic techniques popular at the time. We were encouraged by the earlier experiences of a group in SSL, which had developed a character generator for a similar, but higher resolution display. The display resolution was 606 pixels horizontally by 808 pixels vertically, which allowed display of a full page of text. The display image was refreshed directly from main memory, so arbitrary graphics could also be produced using the machine's load and store instructions. Initially, a specialized instruction was provided to paint characters from a font in memory into the bitmap; this facility was later superseded by the more general BitBlt primitive invented by Dan Ingalls [17]. The Alto contained no support for other graphic primitives, since we were primarily interested in text and engineering drawing applications that could be done with specialized character sets and straight lines.

The decision to provide a high-performance display came directly from our view that the most important purpose of the machine and the software that would be written for it was to provide a high bandwidth interface to the human user. Timesharing systems had made computing more accessible and decreased its

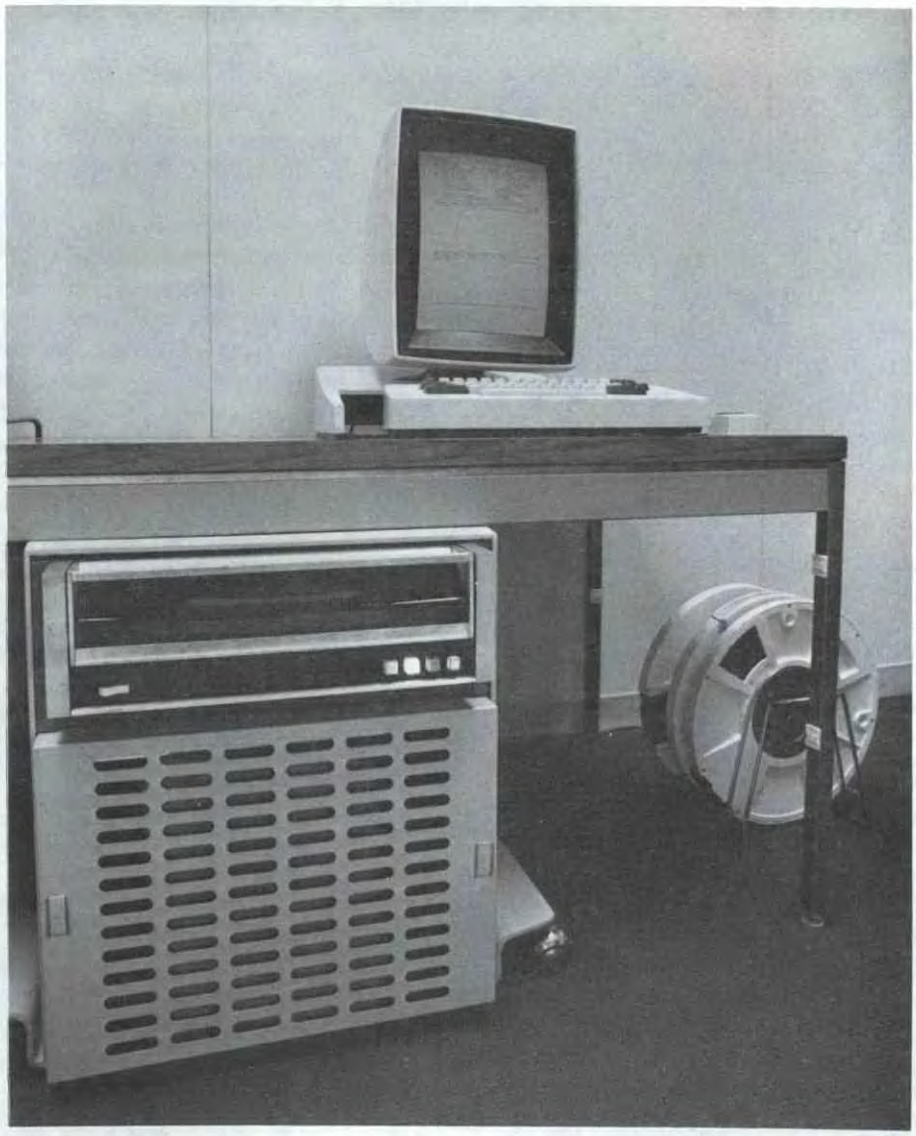


Figure 1: The Alto II Workstation. The Alto I had an identical display, keyboard, and mouse, but a slightly different cabinet.

cost, but they had done little to increase the *quality* of man-machine interaction. We viewed improvement of the user interface as extremely important, and were willing to expend a considerable fraction of the machine's resources in providing it.

Another important feature of the user interface provided by the Alto was the use of a mouse as a pointing device. This was not a PARC innovation; it had been used with considerable success in the pioneering NLS system of Englebart done at SRI in the late sixties [12,11,10]. When rolled over a work surface adjacent to the keyboard, the mouse provides relative positioning information, usually used by software to control the position of a cursor on the display screen. It also provides additional input through buttons on its top surface. Subsequent research [5] has shown that the mouse is a Fitts' Law device, in that it is as efficient for target selection as manual pointing. The practical impact of this is that in the domain for which it was intended, the mouse, iike the compact disk in the audio domain, does as well as the limits of the human user allow. This is often overlooked by those attempting to provide better pointing devices. The mouse is not as effective as a pencil or a graphics tablet for freehand drawing, but very few graphic applications made use of a tablet, although an interface for one was provided.

The display's cursor was a small image, sixteen pix-

els square, whose contents and position could be controlled by software. Many programs made considerable use of the programmability of the cursor, using its contents to convey information about the item to which the user was pointing.

The Alto keyboard was similar to that of a typewriter; it was not accidental that it lacked the cursor positioning keys and numeric keypad found on most personal computers today. In addition to the normal typing keys, it provided eight uncommitted keys that could be used by software as option or function keys. A five-finger keyset, which had been used successfully in Englebart's NLS, was provided as an enhancement to the keyboard, but it required a trained operator for use, and never became popular as an input device.

The original Alto contained 128 thousand bytes of main storage, and a 2.5 million byte cartridge disk. This was similar to contemporary minicomputers, and constituted a fairly serious error. If we had understood how rapidly semiconductor technology would advance, and how long the Alto would live, we would have included more convenient means for accessing a larger memory. We failed to do this, and although the main memory was subsequently expanded to 512 thousand bytes, it was difficult for programs to make use of the additional memory. This hampered software development quite a bit in later years.

The processor of the Alto was specified with flexibility and expansion in mind. It was microcoded, which allowed us to experiment with new instruction sets and with with new input-output devices. The principal characteristic that served to differentiate it from the minicomputers of its time was that the microprogrammed processor was shared between the emulation of a target instruction set and the servicing of up to fifteen additional fixed-priority *tasks*, most of which were associated with the machine's input-output devices. Task switching occurred rapidly, typically every few microseconds. This mechanism allowed the input-output controllers to be very simple, since they could make use of the processor for much of their work. Since access to the single-ported memory is the bottleneck in a small system, multiplexing the processor in this way did not degrade system performance. This technique had been used before on the Lincoln Laboratory TX-2 [14]; it was very successful in the Alto, and has been used in several of the Alto's successors.

To be an effective replacement for centralized computing facilities, personal workstations require a means for communicating with other nearby workstations and with servers that provide shared facilities such as file storage, printing, and long-haul communication. When the Alto design was started, we realized that such a facility would be needed, but did not understand its requirements well enough to begin a design. During late 1972 and early 1973, a number of alternatives were considered, ranging from star-connected serial links operating at a few hundred thousand bits per second to a parallel bus scheme operating at several million bytes per second. The need for bulk file transfers ruled out the low bandwidth of the first approach, and the complexity of the required cabling made the parallel bus unattractive. Coaxial cable, connected with standard cable television components, was tentatively selected as the transmission medium, since it would meet both the bandwidth and reliability requirements.

Several transmission methods were also considered, and a variant of the packet-based Aloha [1] radio system was selected. In pure Aloha, stations needing to transmit packets simply do so, and the resulting interference between stations reduces the channel capacity considerably. We realized that better performance was possible, since our cable Aloha stations could detect when their own transmissions were being interfered with, and abort them without transmitting a complete packet. We chose a baseband, as opposed to a carrier system because it is considerably simpler, and the extra bandwidth afforded by the latter scheme did not appear to be needed. These tentative decisions about the form that the network would take were made in late 1972, but little progress was made on an actual network design until Bob Metcalfe, who had joined CSL in mid-1972, and David Boggs, who came to SSL in March, 1973, began working on what was to become the Ethernet [31].

3 Implementation

In November 1972, implementation of the Alto began. The design was completed in approximately two months, including an initial version of the microcode for an instruction set emulator and for the display and disk controllers. Two prototype machines were then built using wirewrap technology, and were in operation in April 1973. One reason for the short schedule was that we had developed a rapid prototyping facility as a part of the construction of the Maxc timesharing system during 1971 and 1972. We were also able to use the memory boards originally developed for the Maxc system in the Alto, which saved considerable engineering effort. The design of the processor, memory subsystem, and display controller was done by Chuck Thacker, while the disk controller and its microcode were designed and im-

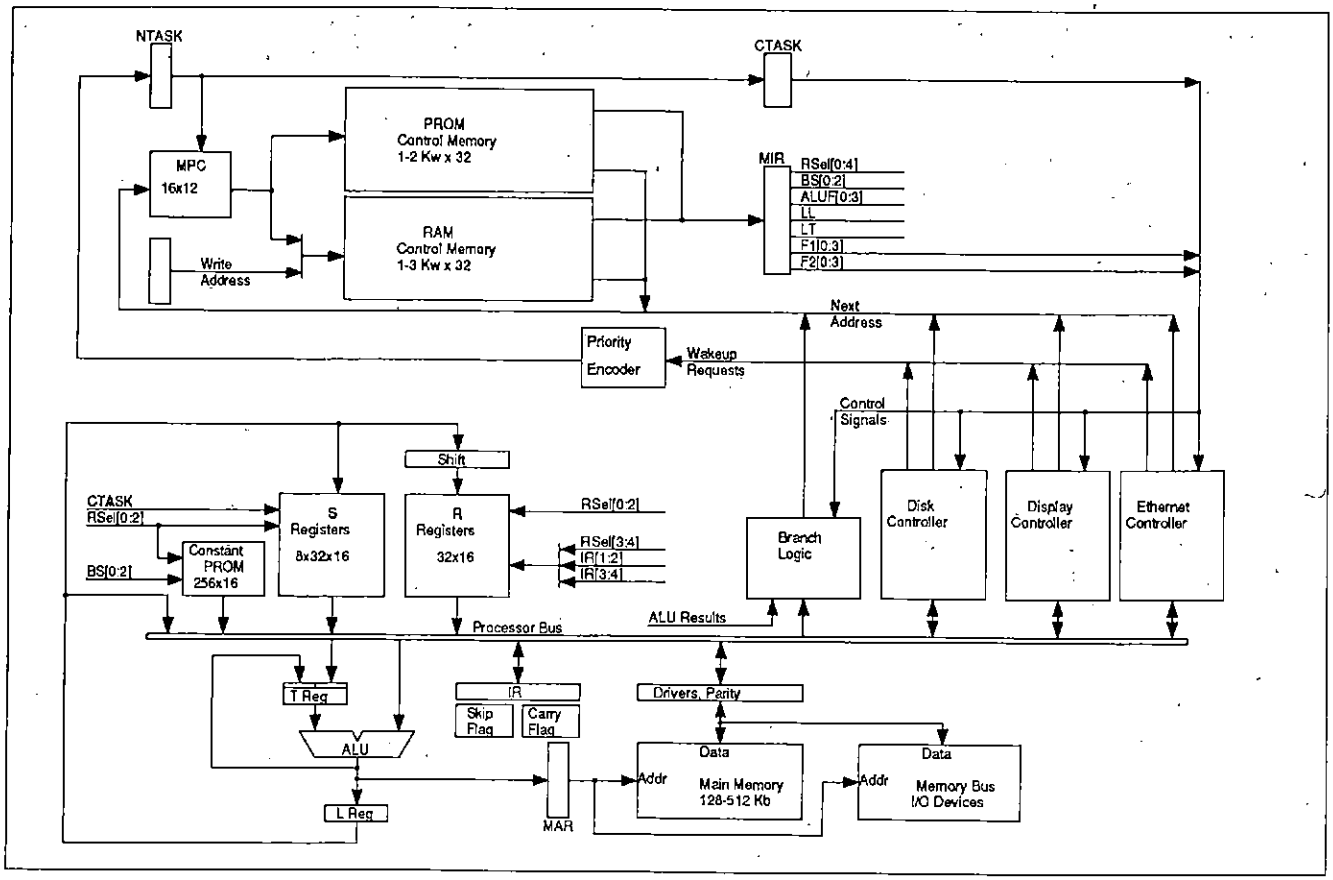


Figure 2: The Alto Microprocessor.

plemented by Ed McCreight. Larry Clark built the early systems and designed the package.

The Alto was a very simple machine by today's standards. The processor (Figure 2), is composed of three printed circuit boards containing about 200 small and medium-scale integrated circuits. Each of the input-output controllers occupies a single board containing approximately sixty integrated circuits. The processor is organized around a 16-bit bus connected to the main memory, an arithmetic unit, a number of high speed registers (R and S), and the input-output controllers. The transfer of data over the bus, the operations to be performed on the data, and the tests to be applied to it are controlled by a 32-bit microinstruction taken from PROM or RAM control store. Microinstructions are executed by a two stage pipeline which can start a new instruction every 170 nanoseconds. The processor is shared among sixteen fixed priority tasks. The NTASK and CTASK registers hold the number of the task currently in control of the processor. NTASK addresses the sixteen-element MPC RAM, which holds the task program counters. NTASK is loaded with the number of the highest priority wakeup request whenever the running task is willing to relinquish the processor. The only state associated with a task that is saved by the hardware is the task's program counter. Other machine resources are shared among tasks by programming convention. Normally, a task switch takes place with no overhead, unless it is necessary for the task giving up control to save and restore the L or T register.

The Alto main memory is synchronous with the processor, which starts all references by explicitly loading the memory address register (MAR). The memory can read or write a single 16-bit word in five machine cycles, or it can read a 32-bit doubleword in six cycles. The doubleword read was originally provided to support the display, which consumes two-thirds of the memory bandwidth even with this operation; it was also used very effectively by instruction set emulators for instruction fetching and to manipulate 32-bit quantities.

The Alto was not a high-performance machine, even by the standards of its time. Without the performance degradation caused by the input-output devices, it requires between one and three microseconds to execute a single emulated instruction. With the display running, these times are increased by a factor

of three. Until software was developed that required a great deal of computation for simple user actions (e.g., the Bravo text editor), the speed of the machine was adequate. Perhaps more important than the absolute speed is the fact that the performance of the Alto is *predictable*. It is very difficult to provide this characteristic in a timesharing system, and its lack can be very disconcerting to the user. The Alto cannot provide the peak performance of a timesharing machine, but it has the desirable property, pointed out by Jim Morris, that it doesn't run faster at night.

Once the design was complete, the microprocessor was simple enough that the hardware worked almost immediately. Debugging the microcode was more difficult, but was simplified considerably by an auxiliary writeable microstore built for the purpose. This device was connected to the control logic of the Alto under test. It replaced the PROM control store with RAM, and also added several bits to the microword. These additional bits were used to provide a breakpoint capability that made debugging much easier. The test unit was under control of a dedicated minicomputer that ran a microcode assembler and debugger. Using these tools, microcode debugging could be carried out as easily as debugging an assembly language program on a conventional machine.

The first of several microcoded instruction set emulators developed for the Alto was done for a virtual machine similar to the Data General Nova minicomputer. We had previously purchased a number of these systems, and had developed for them a compiler for BCPL [32], a predecessor of the popular C language. The main differences between the Alto instruction set (Figure 3) and that of the Nova were that the size of the Alto's address space was twice that of the Nova, and a number of instructions were added to support the Alto's input-output and interrupt system and to optimize BCPL procedure calls.

The resulting instruction set was not compatible with that of the Nova, but it was sufficiently similar that modifying the compiler was straightforward. Most of the early software for the Alto was written in BCPL. Only a small amount of assembly code was ever written for the machine. The microprocessor hardware contained a small amount of logic to enhance the performance of the BCPL emulator. This logic included a register to hold the instruction being executed and a method to address the registers used for the emulated machine's accumulators from its fields. This logic was not used by later emulators, and it probably would have been better to have used the same amount of hardware to provide functions with more general utility.

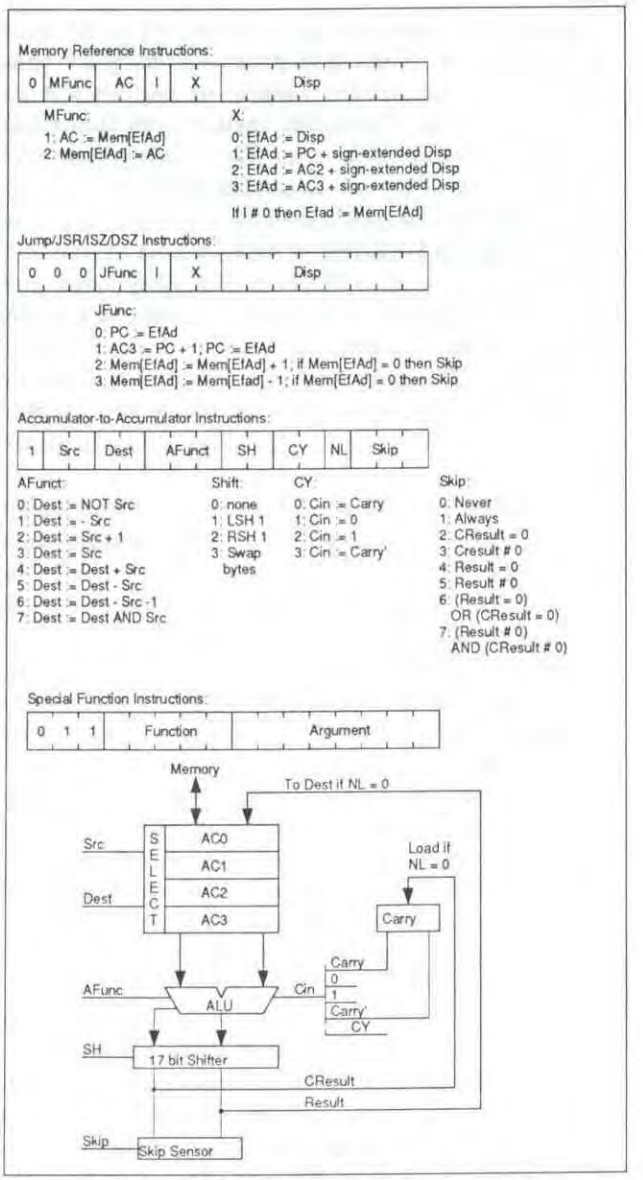


Figure 3: The BCPL Instruction Set and Processor Model

Emulators for several other instruction sets, including Smalltalk [18,16], Lisp [8,7], and Mesa [15] were written for the Alto. All were based on the idea of encoding the instructions as *bytecodes*. This arrangement allows instruction decoding to be done by a single 8-bit dispatch, rather than the several dispatches required to decode a BCPL instruction. The performance gained in this way more than offsets the lack of specialized decoding hardware. Most of the software written for the Alto after 1977 used Mesa, which was the choice in CSL, or Smalltalk, which was the language of choice in LRG. The Alto Lisp system was unsuccessful, primarily due to the lack of sufficient memory.

Input-output devices can be connected to the Alto in several ways. High performance controllers that make use of the microprocessor or require a high bandwidth connection to the memory are connected directly to the processor bus. The display, disk, and Ethernet controllers are examples of devices in this class. Lower performance devices are attached to the memory bus and addressed as locations at the end of memory, as in the PDP-11. The keyboard and keyset are connected in this way. Finally, a parallel port is provided for low performance devices outside the cabinet. This port was used to connect a variety of devices, including impact printers, a PROM programmer, and X-Y digitizing tablet, and a cassette tape deck.

The hardware controllers for the disk, the display, and the Ethernet are similar. They contain data buffering, logic to drive and receive control lines required by the device, and a small amount of device-specific timing and control logic. Most of the complexity associated with a controller is contained in its microcode. The display, for example, makes use of three of the Alto's sixteen microtasks: one is awakened during the display's vertical synchronization period, one is awakened during horizontal synchronization, and one is responsible for transferring raster data from main memory to the controller data buffer. The controller hardware provides wakeup requests at the appropriate times, but the microcode is responsible for carrying out most of the work required to maintain the display.

This arrangement, in which a device controller has the full computational power of the processor at its disposal, allowed us to provide convenient logical interfaces between devices and the driver software that operated them. The display interprets a chain of control blocks in main memory, each of which defines the contents of a horizontal band on the display screen. Since areas of the screen containing only white space, such as the space between lines and paragraphs, are not required to have any underlying bitmap memory, this technique reduces the display storage required by the standard text editor from the 61 thousand bytes required to represent a full screen to approximately 50 thousand bytes, a substantial saving in a machine with only 128 thousand bytes of memory. Drawing programs that allow the user to manipulate full-screen images cannot take advantage of this economy, but such programs are considerably simpler and smaller than the editor, so efficient use of space is less important in these applications.

The disk controller also makes extensive use of the Alto microprocessor. Like the display, it executes a chain of control blocks from main memory. Each control block specifies a 512 byte page to be read or written. The controller is able to transfer consecutive physical sectors between the disk and memory, which represented unusually high performance for the time. The controller uses two of the Alto's microtasks, one of which is awakened every sector, and one which is responsible for data transfers within a sector.

The disk controller and the file system were designed concurrently by Ed McCreight and Butler Lampson, with reliability as a primary goal. The designers wanted to provide a system in which hardware or software errors would result in a minimum amount of information loss. An important innovation—the use of label blocks—contributed substantially to the system's reliability. Labels add a third record to the header and data records customarily contained in a disk sector. The label record contains a unique integer identifier for the file containing the sector, the sector's position in the file, the addresses of the previous and next pages of the file, and the number of valid bytes in the data record. The controller microcode checks the information in the label block before doing any operation on the data record. This check ensures that the disk is properly positioned, both physically and logically, before any access to the data is done.

The use of label blocks, combined with replication of directory information in the first page of every file, means that the directory, which is an ordinary file, can be reconstructed from the contents of the disk itself if it becomes scrambled. Similarly, if data pages are corrupted, it is possible to determine precisely the extent of the loss, and to preserve the balance of the file system. One of the earliest programs written for the Alto was the Scavenger, which verified the integrity of a file system and corrected inconsistencies. This program, the first version of which was written by Jim Morris, makes the loss of even a small portion of a file system an extremely rare event.

The original Alto prototypes did not contain Ethernet interfaces, but during 1973, substantial progress on both network hardware and protocols was made. The name "Ethernet" was first used in May 1973 by Bob Metcalfe. Metcalfe and David Boggs worked on the network facilities during the summer and fall, and the prototype machines were exchanging packets by the end of 1973. Although the Alto was the first machine equipped with an Ethernet interface, Metcalfe and Boggs went on to design controllers for all the PARC computers, including Data General Novas and DEC PDP-11s. The Ethernet transceiver, an analog device that connects the controller to the coaxial cable, was designed by Tat C. Lam. The original Ethernet was slower than the commercial version available today. Its transmission rate was 2.94 million bits per

second, half the rate of the Alto master clock, and it used Manchester encoding for the serial data. For collision detection, it relied on comparison between the actual signal on the coaxial cable and the signal the transmitting station was attempting to send, rather than the level monitoring done by the commercial version. The 3 million bit per second bandwidth of the network was of some concern initially, in that we were not sure that it would be sufficient for a large system. Subsequent measurements [33] of a large network revealed that loads in excess of ten percent of full capacity were rare. There might have been less excess capacity if diskless workstations that paged over the network had been employed, but the use of local disks made this unnecessary.

In April, 1973, the first Alto prototype was completed. It was able to run simple programs to exercise the disk and display. The first image to be displayed was the Sesame Street 'Cookie Monster', which had been carefully digitized by a member of Kay's group.

During the balance of 1973, nine more prototype machines were built at PARC. During the summer, the prototypes had been tested sufficiently that we were willing to commit the design to printed circuit boards. At the same time, we realized that the lack of writeable control store would be a serious limitation in a machine intended for experimentation, and added one thousand words of instruction RAM to the original PROM control store. The original microcode was improved substantially, and a number of test programs for the hardware were written. Software development had begun, but there was not enough software available by the end of 1973 for the Alto to replace our timesharing system as the main computing facility for users.

One of the major strengths of the Alto in a research environment was that it could provide very high performance if the user were willing to accept the unpleasantness of microprogramming the processor directly. The first demonstration of such an application took place in the summer of 1973. Alan Kay, who was an accomplished organist, wanted a synthetic instrument with natural sound quality. He believed that this could be accomplished by recording and digitizing a real organ, and doing table lookup to provide a completely faithful replica of the waveshape. We decided to give him his wish, and purchased a two-manual organ keyboard (with pedals), a precision digital-to-analog converter, and a high fidelity amplifier and loudspeaker system. The keyboard and DAC were interfaced so that a task-specific function could read the keyboard as a bit vector, and load a value into the DAC. The organ simulator kept in main memory a table, consisting of the amplitudes at 256 points along a single cycle of the desired waveshape. Samples were generated for the DAC at a 25 kHz rate by taking points from this table at offsets which were inversely related to the frequency to be generated. The microprogram computed up to ten such samples every 40 microseconds, one for each key that was depressed. Samples were summed and the result was transmitted to the DAC. A variety of different effects could be generated by using different variations of the 'canonical cycle'. Although this application was fairly frivolous, it was an impressive demonstration of the real-time capability of the machine. Later, a serial line concentrator that connected up to sixty-four serial 300 baud lines to the Ethernet was built using similar techniques. The only extra hardware used in this device was a group of level converters and latches to allow the microprogram to read the value of the received data and store the data to be transmitted. All other processing was done by specialized microcode.

By late 1973, we were ready to produce a quantity of the machines for CSL and SSL. Although we had been able to build a small number of prototypes, the manufacture of the thirty systems we needed was beyond our capabilities. Fortunately, the company had established a custom systems manufacturing group in Los Angeles, and this group agreed to manufacture the Alto for us. The first machines were delivered between May and September, 1974. This was only slightly later than the first release of the basic Alto software, which took place in March, 1974. This early software consisted of the operating system [28], the BCPL compiler, and a primitive text editor. It allowed many of the researchers in CSL and SSL to begin doing a substantial amount of their work on the Alto, although the Maxc timesharing system was still used for electronic mail, file storage, and printing.

4 Servers

The most important components of a distributed computing system, after the workstations and the network that interconnects them, are the servers that provide shared facilities. We initially underestimated the importance of servers, assuming that the facilities provided by a set of workstations would be sufficient. We soon discovered that this was incorrect. Some functions, such as high-quality printing, are very expensive, and must be shared for economic reasons, while in other cases, sharing is used to provide communication between the users of the system. File storage is an example of the latter situation, although the low cost per byte of large disk files also provides economy of scale.

The first PARC server was EARS, a printing server named after its components: Ethernet, Alto, RCG (research character generator), and SLOT (scanning laser output terminal). This system was quite successful, in spite of the considerable obstacle to its acceptance presented by its bizarre name. It was the forerunner of the Xerox 9700 printer, which has been an extremely successful product.

The printing portion of EARS had been under development even before PARC was founded. Gary Starkweather, an optical engineer at the Xerox Webster Research Center, joined SSL in 1971. He brought with him a prototype printing engine consisting of a laser scanner attached to a standard Xerox copier. This device used a rotating polygon to scan an intensity-modulated laser beam across a standard xerographic drum, building up a raster image of the page being printed. During 1972, Ron Rider of SSL and Butler Lampson designed and implemented a character generator capable of printing high quality text in several fonts on Starkweather's engine, at a speed of one page per second.

A printer based on these components, driven by a Data General Nova minicomputer, was demonstrated in late 1973, but was never placed in service. Instead, Rider decided to build an Ethernet-based print server. He modified the character generator to allow it to be driven from an Alto, and wrote the necessary software to control the printer. The network-related portions of the server were written by Bob Metcalfe. EARS was the first major application for the Ethernet, and during its development, several experimental communications protocols [30] were tested and refined. EARS was placed in service in the Fall of 1974, and provided printing service to CSL and SSL until it was replaced by the Dover printer in 1977.

The final component needed for a complete distributed computing system was a file server. The hardware basis for several experimental file servers was a high-performance disk controller designed by Roger Bates in 1975. Like earlier Alto device controllers, it made use of the Alto's microprogrammed tasking for many of its functions, but it was considerably more complex than the earlier controllers because of the high bandwidth of the attached disks. Using this device, it was possible to connect as many as seven 300 million byte disks to a single Alto.

The first file server, Juniper, was to have provided page-level access to files, as well as atomic transactions. Planning for Juniper began in 1974, but actual programming did not begin until 1976. It was completed in 1977, but was never used extensively because of its poor performance.

The file server that was used most widely was the "Interim File Server", or IFS. This software was written in 1976 by Ed Taft and David Boggs, when it became clear that Juniper would not be ready as early as originally anticipated. It was an extension of the simple Alto file system, combined with the PUP (PARC Universal Packet) file transfer protocol [3]. It provided only bulk file storage, but it was completed rapidly and was reliable and efficient. However, as an "interim" system, it was a failure, since the IFS still provides the majority of the file storage in the Xerox internetwork.

The most complex server built using the Alto was the Dover printer, designed in 1976. Dover used a raster printing engine that was a descendant of the SLOT used in the EARS server, driven by a controller that relied heavily on the input-output processing capability of the Alto. Dover was a large project, involving several groups within the company. The design of the printing engine was done by the Special Projects Group (SPG) in Los Angeles, the group that manufactured the Alto. John Ellenby of CSL was responsible for management of the engine development. Software for the Dover was written by Dan Swinehart of SSL, and Bob Sproull of CSL. The development of the controller was done by Severo Ornstein of CSL, Bob Sproull, and Jim Leung of SPG, from a design by Butler Lampson.

The controller was considerably simpler than the earlier EARS character generator. Instead of using hardware to generate the bit stream for the printer in real time, the controller built up the image to be printed incrementally. Two buffer memories were used, each capable of holding sixteen of the printer's scan lines. While one of these buffers was being serialized and sent to the printer under hardware control, the other was being loaded with video information. The text to be printed and the bitmap representations for the fonts to be used were kept in Alto main memory, and written into the buffer by a high priority microcoded task. This process was then repeated for each of the roughly 250 bands that made up each page. Spooling of files received from the network was done by a BCPL program, which also sorted the contents of each page into bands in preparation for transmission to the controller. The resolution of the Dover was 384 pixels per inch, lower than the 500 pixels per inch of EARS, but still adequate for text and line graphics. The bandwidth requirements of the printer were high enough that the machine could not receive and spool files while printing, but its printing speed of one page per second was high enough that this was not a problem, since it could stop between pages to receive files. Incremental image generation also placed a limit on the complexity of pages that could

be printed. For the few images that exceeded the limit, another server that composed an entire page on disk and transmitted it to a slower printing engine was available.

The controller contained approximately 300 integrated circuits, making it about one eighth as complex as the character generator used in EARS. This simplification was made possible by the extensive use of the Alto's microprocessor to provide the low level control functions for the interface. Several dozen Dovers were built, and a number are still in operation.

5 Expansion

During 1975, the Alto was redesigned to improve its reliability and reduce its cost. The work was done by the Special Projects Group in Los Angeles that had been producing Altos for PARC, and was planned and supervised by John Ellenby of CSL. As part of this effort, all the boards and the package were redesigned. The memory system was reimplemented using 4 thousand bit RAM chips, and error correction was added. The resulting machine was much easier to build and service than the earlier system, and its cost was much lower—approximately twelve thousand dollars, rather than eighteen thousand dollars. The Los Angeles group had built a total of sixty of the original systems over a period of two years, most of which had been purchased by CSL and SSL. They were now able to produce the system in high volume. Over the next four years, approximately fifteen hundred Altos were built, of which approximately a thousand are still in use today. Most are used by individual engineers and scientists in a number of Xerox facilities, although many were configured as servers, and a few were used for marketing probes or donated to university computer science groups.

A final redesign done in 1979 replaced the memory with sixteen thousand bit RAM chips, and increased the amount of memory that could be attached to 512 thousand bytes. At the same time, the microcode store was changed from one thousand words of RAM and two thousand words of PROM to one thousand words of PROM plus three thousand words of RAM. By this time, other language emulators had almost totally superseded the original BCPL emulator. These emulators were usually loaded at bootstrap time or as part of starting a program that used a particular language. The additional RAM control store made it possible to spend less time minimizing the space required by the microcode and concentrate instead on its functionality and performance. The microcode was improved somewhat after the final redesign, but by 1980, most new development had shifted to the Alto's successors, described in the next section.

Although most of the software developed in CSL and SSL was distributed with the Alto, three application programs were primarily responsible for the machine's popularity with technical professionals in Xerox. The Bravo editor [22], designed by Charles Simonyi and Butler Lampson, and implemented and improved by Simonyi and others from 1974 through 1978, was the first and most important of these. Bravo was the first WYSIWYG (what you see is what you get) editor. It supported multiple fonts, and its high quality output could be printed on one of the many Dover printers that were available throughout the company.

The second important application program was the Laurel mail system [4], written by Doug Brotz, Roy Levin, and Mike Schroeder in 1978. Electronic mail has a profound effect on communication within an organization, since it combines the permanence and precision of memos with the speed of the telephone. By 1980, the Xerox internetwork, composed of local Ethernets connected by telephone lines, had been expanded to most of the engineering and research sites within the company. Laurel was rapidly adopted by a large fraction of this community, and by 1983 there were over four thousand regular electronic mail users in Xerox.

The third popular application was a group of tools for digital logic design, including the SIL illustrator [35] written by Chuck Thacker in 1975, and a routing program for prototypes written by Ed McCreight. These programs also produced high quality documentation on the Dover printers. They increased the productivity of designers significantly, and are used by most of the electronic engineers in the company.

6 Successors

The Alto was only the first of several personal workstations built at Xerox. The Dorado [26,6,27] and the Dicentra were developed at PARC, and the Dolphin and the Dandelion were commercial systems designed in the Electronics Division and the System Development Division.

Dorado is the largest hardware engineering project ever undertaken by the Computer Science Laboratory. It was difficult to think of the Dorado as a personal machine, since it consumed 2500 watts of power, was the size of a refrigerator, and required 2000 cubic feet of cooling air per minute (while producing a noise level that has been compared to that

of a 747 taking off).

It was *used* as a personal machine, however, and supplied computing power comparable to three VAX-11/780s. This may seem profligate, but it was consistent with the view that the CSL hardware base should be equivalent to that which would be commercially available and affordable in five to ten years. With hardware that is not limiting, it is possible to explore ways of using computers that are considerably ahead of current practice.

The Dorado project was started in CSL during 1975. It was moved to the System Development Division in 1976, but returned to CSL in 1977 when it became clear that the machine's high cost would not meet SDD's needs. The initial design was completed in late 1978, and two prototypes were built. A redesign, completed in 1979, was then done to simplify the machine. Manufacture of the machine started in 1980 in a small production facility that had been established for the purpose. By 1982, approximately thirty systems had been built, and the Dorado had replaced the Alto as the principal computing vehicle in CSL and SSL.

The Dorado achieved its high performance through its aggressive technology and a great deal of attention to detail on the part of its designers. It uses emitter-coupled logic (MECL 10K) with two to four nanosecond gate delays. The processor is microprogrammed, and like the Alto employs multitasking at the microcode level to operate input-output controllers. Unlike the Alto, it has virtual memory, an eight thousand byte cache, and a separate instruction fetch unit associated with the CPU. Up to 16 million bytes of main memory may be attached to the Dorado. The memory bandwidth available for input-output devices and to service cache misses is 66 million bytes per second. The processor executes microinstructions in a three-stage pipeline that can start a new instruction every sixty nanoseconds. The separate instruction fetch unit allows many instructions in the common emulators to execute in a single microinstruction.

In terms of man-years expended on a single project in CSL, the Dorado is second only to the Cedar programming environment [9,34], which its high performance made possible. The initial design was done by Butler Lampson and Chuck Thacker; the design was continued in SDD by Thacker, Brian Rosen, Don Charnley, and Tom Chang. When the project returned to PARC, it was partitioned into a number of subsystems: Ed McCreight and Severo Ornstein were the project managers; Butler Lampson was the technical leader of the project. The microprocessor was designed by Ken Pier, Roger Bates, and Ed Fiala. The instruction fetch unit was designed and implemented by Severo Ornstein, Gene McDaniel, and Will Crowther. The storage system was done by Doug Clark, Ed McCreight, and Ken Pier. A number of individuals produced the microcode for the machine, including Ed Taft, Peter Deutsch, Willie-Sue Haugeland, and Nori Suzuki.

The Dolphin was a much less ambitious successor to the Alto, designed in the Electronics Division of Xerox in 1977-79. In a sense, it was the successor of the Dorado, rather than the Alto, since it was done by the same group (Thacker, Charnley, Rosen, Chang) that had worked on the Dorado in SDD as well as a group in Los Angeles that included Jack Cameron, Howard Kakita, and Malcolm Thomson. Dolphin employed a number of ideas that had been incorporated into the Dorado, including virtual memory and a high-bandwidth input-output system. Its technology was not as aggressive as that of the Dorado—Schottky TTL, rather than ECL— and it was smaller and much less expensive. The Dolphin was used as the processor in the Xerox 5700 Electronic Printing System, and a version configured as a Lisp workstation became the 1100 Scientific Information Processor. Although PARC built approximately fifty Dolphins for internal use, and provided emulators for the Alto instruction set, Mesa, and Lisp, the machine was not popular. It had a higher resolution display, more memory, and a larger disk than the Alto, but it was only about twice as fast. Dolphin became available slightly before the Dorado, but the performance of the latter machine made it much more attractive, particularly for a research environment.

The Dandelion [19], known commercially as the Star 8010 workstation, was implemented in 1979 and 1980 by a group in SDD consisting of Bob Belleville, Robert Garner, and Ron Crane. Dandelion was based on a paper design called Wildflower done by Butler Lampson and Roy Levin of CSL. It was intended to have high performance and extremely low cost, but limited configuration flexibility. The Dandelion CPU used 2901 bit-slice processors, and employed a fixed time-slice form of multitasking that was quite different from that of the Alto. Dandelion was the first of the Alto's descendants that did not provide an emulation mode in which Alto software could be run. It was programmed exclusively in Mesa and the extended Mesa developed for the Cedar system. Dicentra, built by David Boggs and Hal Murray in 1982, was a variant of the Dandelion which included a Multibus rather than a proprietary bus for the attachment of input-output devices and memory. It provided a low-cost way to obtain a Mesa-compatible processor to which industry standard peripheral controllers and devices could be attached.

In addition to its direct descendants at Xerox, the Alto has inspired a number of similar systems from other commercial vendors. The Apple Lisa and Macintosh are perhaps the most familiar of these; Table 3 in Lampson [24] lists several others.

7 Conclusion

The Alto is small and slow by today's standards. The four generations of memory and microprocessor development that have passed since 1972 have made it straightforward to build low cost personal workstations with a hundred times the memory capacity and ten times the speed of the Alto. It seems likely that progress in semiconductors will continue at its present rate for perhaps another decade before fundamental physical limits are reached, so much more powerful systems are inevitable.

Higher bandwidth networks have also become much easier to engineer with the advent of fiber optics. However, experience with the Ethernet indicates that even with very high performance machines such as the Dorado, network bandwidth is not the limiting factor in overall system performance.

A surprising fact that has emerged from the work on the Alto and its successors is that the amount of software required to support interactive user interfaces is much greater than originally anticipated. Invariably, the complexity of the software is much greater than that of the hardware on which it runs. Except in a few applications in which the users are experts (e.g., programmers using programming environments), it has not yet become possible to provide the kind of symbiotic relationship between computer and human envisioned by Licklider in 1960. Advances in programming technology, as well as better hardware, will be required to achieve the kind of system he described. Distributed personal computing systems will help bring about these advances by providing more productive and efficient computing environments for developers.

References

[1] N. Abramson. The ALOHA System. In *Proc. AFIPS FJCC*, pages 281–285, 1970.

[2] D.G. Bobrow et al. Tenex: A paged time-sharing system for the PDP-10. *Communications of the ACM*, 15(3):135–143, March 1972.

[3] D.R. Boggs et al. Pup: An internetwork architecture. *IEEE Trans. Comm.*, 28(4):612–624, April 1980.

[4] D.K. Brotz. *Laurel Manual.* Technical Report CSL-81-6, Xerox Palo Alto Research Center, 1981.

[5] S. Card et al. Evaluation of mouse, rate-controlled isometric joystick, step keys and text keys for text selection on a CRT. *Ergonomics*, 21(8):601–613, August 1978.

[6] Douglas W. Clark, B.W. Lampson, and Kenneth A. Pier. The memory system of a high-performance personal computer. In *The Dorado: A High-Performance Personal Computer—Three Papers, CSL-81-1*, pages 51–80, Xerox Palo Alto Research Center, 1981.

[7] L.P. Deutsch. Experience with a microprogrammed Interlisp system. *IEEE Transactions on Computers*, C-28(10), October 1979.

[8] L.P. Deutsch. A Lisp machine with very compact programs. In *Proc. 3rd IJCAI*, Stanford, 1973.

[9] L.P. Deutsch and E.A. Taft. *Requirements for an experimental programming environment.* Technical Report CSL-80-10, Xerox Palo Alto Research Center, June 1980.

[10] D.C. Engelbart. The augmented knowledge workshop. In *Proc. ACM Conf. on History of Personal Workstations*, January 1986.

[11] D.C. Engelbart. A conceptual framework for the augmentation of man's intellect. In Howerton and Weeks, editors, *Vistas in Information Handling, volume 1*, pages 1–29, Spartan Books, Washington, 1963.

[12] D.C. Engelbart and W.K English. A research center for augmenting human intellect. In *Proc. AFIPS Conf.*, pages 395–410, 1968.

[13] E.R. Fiala. The MAXC systems. *IEEE Computer*, 11(5):57–67, May 1978.

[14] J.W. Forgie. The Lincoln TX-2 input-output system. In *Proc. Western Joint Computer Conf.*, pages 156–160, February 1957.

[15] C.M. Geschke et al. Early experience with Mesa. *Communications of the ACM*, 20(8):540–553, August 1977.

[16] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[17] D. Ingalls. The Smalltalk graphics kernel. *Byte*, 6(8):168–194, August 1981.

[18] D.H. Ingalls. The Smalltalk-76 programming system: Design and implementation. In *Proc. 5th ACM Symp. Principles of Prog. Lang.*, pages 9–16, January 1978.

[19] R.K. Johnsson and J.D. Wick. An overview of the Mesa processor architecture. *ACM Sigplan Notices*, 17(4):20–29, April 1982.

[20] A.C. Kay. Microelectronics and the personal computer. *Scientific American*, 237(3):236–245, September 1977.

[21] A.C. Kay. *The Reactive Engine.* PhD thesis, University of Utah, 1969.

[22] B.W. Lampson, editor. *Alto User's Handbook.* Xerox Palo Alto Research Center, 1976.

[23] B.W. Lampson. Guest editorial. *Software–Practice and Experience*, 2:195–196, 1972.

[24] B.W. Lampson. Personal distributed computing: The Alto and Ethernet software. In *ACM Conf History of Personal Workstations*, January 1986.

[25] B.W. Lampson et al. A user machine in a time-sharing system. *Proc. IEEE*, 54(12):1744–1766, December 1966.

[26] B.W. Lampson, Gene A. McDaniel, and Severo M. Ornstein. An instruction fetch unit for a high-performance personal computer. In *The Dorado: A High-Performance Personal Computer—Three Papers, CSL-81-1*, pages 21–50, Xerox Palo Alto Research Center, 1981.

[27] B.W. Lampson and K.A. Pier. A processor for a high-performance personal computer. In *Proc. 7th Symp. Computer Arch.*, pages 146–160, ACM Sigarch/IEEE, May 1980.

[28] B.W. Lampson and R.F. Sproull. An open operating system for a single-user machine. *ACM Operating Sys. Rev.*, 13(5), November 1979.

[29] J. Licklider. Man-computer symbiosis. *IRE Trans. Human Factors in Electronics*, HFE-1:4–11, March 1960.

[30] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[31] R.M. Metcalfe, D.R. Boggs, C.P. Thacker, and B.W. Lampson. U.S. Patent 4,063,220: Multipoint Data Communication System With Collision Detection. December 1977.

[32] M. Richards. BCPL: A tool for compiler writing and system programming. In *AFIPS Conf. Proc.*, pages 557–566, 1969.

[33] J.F. Shoch and J.A. Hupp. Measured performance of an Ethernet local network. *Communications of the ACM*, 23(12):711–721, December 1980.

[34] W. Teitelman. A tour through Cedar. *IEEE Software*, 1(4), April 1984.

[35] C.P. Thacker. SIL—a simple illustrator for cad. In S. Chang, editor, *Fundamentals Handbook of Electrical Computer Engineering, Volume 3*, pages 477–489, Wiley, 1983.

[36] C.P. Thacker et al. Alto: A personal computer. In Siewiorek et al., editors, *Computer Structures: Principles and Examples*, chapter 33, McGraw-Hill, 1982. Also CSL-79-11, Xerox Palo Alto Research Center (1979).

Personal Distributed Computing: The Alto and Ethernet Software

Butler W. Lampson
Systems Research Center
Digital Equipment Corp.
130 Lytton Avenue
Palo Alto, CA 94301

Abstract

The personal distributed computing system based on the Alto and the Ethernet was a major effort to make computers help people to think and communicate. The paper describes the complex and diverse collection of software that was built to pursue this goal, ranging from operating systems, programming environments, and communications software to printing and file servers, user interfaces, and applications such as editors, illustrators, and mail systems.

1 Introduction

A substantial computing system based on the Alto [58] was developed at the Xerox Palo Alto Research Center between 1973 and 1978, and considerably extended between 1978 and 1983. The hardware base for this system is simple and uniform, if somewhat unconventional. The software, which gives the system its visible form and knits together its many parts, is by contrast complex, variegated, even ramified. It is convenient to call the whole complex of hardware and software "the Alto system." This paper describes the major software components of the Alto system. It also tries to explain why the system came out as it did, by tracing the ideas that influenced it, the way these ideas evolved into the concept of personal distributed computing, and the nature of the organization that built the Alto. A companion paper by Chuck Thacker [56] describes the hardware.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0101 $00.75

1.1 Themes

"But a man's reach should exceed his grasp, or what's a heaven for?" *Browning*

The Alto system grew from a vision of the possibilities inherent in computing: that computers can be used as tools to help people think and communicate [39]. This vision began with Licklider's dream of man-computer symbiosis [32]. When he became head of the Information Processing Techniques Office at ARPA, Licklider pursued the dream by funding the development of computer time-sharing. His successors, Ivan Sutherland, Robert Taylor, and Larry Roberts, continued this work throughout the 1960's, and extended it to computer networks. Taylor went on to manage the laboratory in which most of the Alto system was built, and people who worked on these ARPA projects formed its core. Their collective experience of time-sharing and networks became a major influence on the Alto design.

Another important influence, also funded by ARPA, was Doug Engelbart. He built a revolutionary system called NLS, a prototype of the electronic office. In NLS all the written material handled by a group of people is held in the computer and instantly accessible on a display which looks something like an ordinary sheet of paper. He demonstrated this system at the 1968 Fall Joint Computer Conference in San Francisco [12,13]. NLS was too expensive to be practical then, but it made a profound impression on many of the people who later developed the Alto.

Yet another ARPA project that had a strong influence on the Alto was Alan Kay's Flex machine, also called the Reactive Engine [21]. Kay was pursuing a different path to Licklider's man-computer symbiosis: the computer's ability to simulate or model any system, any possible world, whose behavior can be precisely defined. And he wanted his machine to be small, cheap, and easy for non-professionals to use. Like Engelbart, he attached great importance

to a high-quality, rapidly-changing display. He later coined the name "Dynabook" for the tool he envisioned, to capture its dynamic quality, its ubiquity, and its comfortable fit with people [22].

The needs of the Xerox corporation also influenced the Alto, since Xerox provided the money that paid for it. In 1970 Xerox started a research center in Palo Alto called PARC to develop the "architecture of information" and establish the technical foundation for electronic office systems which could become products in the 1980's. It seemed likely that copiers would no longer be a high-growth business by that time, and that electronics would begin to have a major effect on office systems, the firm's major business. Xerox was a large and prosperous company with a strong commitment to basic research and a clear need for new technology in this area. To the computer people at PARC, this looked like a favorable environment in which to take the next step toward making computers into effective tools for thinking.

The electronic office and the Dynabook, then, were the two threads that led to the Alto system. The idea of the electronic office is to do all the work of an office using electronic media:

- capturing information,
- viewing it,
- storing and retrieving it,
- communicating it to others, and
- processing it.

The idea of the Dynabook is to make the computer a personal and dynamic medium for handling information, which can model the world and make the model visible.

Of course the Alto system is not an electronic office, or a Dynabook; these were ideals to draw us on, not milestones to be achieved (or millstones either). In the course of developing the Alto we evolved from these ideals to a new style of computing, which is the preferred style in the 1980's just as time-sharing was the preferred style of the 1970's; witness the conference for which this paper was prepared. For this style there is no generally accepted name, but we shall call it "personal distributed computing." Why these words?

The Alto system is *personal* because it is under the control of a person, and serves his needs. Its performance is predictable. It is reliable and available. It is not too hard to use.

And it is fast enough. In 1975 it was hard for people to believe that an entire computer is required to meet the needs of one person. The prevailing attitude was that machines are fast and people are slow; hence the merits of time-sharing, which allows one fast machine to serve many of the slow people. And indeed time-sharing, with response times measured in seconds, is an advance over a system which responds in hours. But this relationship holds only when the people are required to play on the machine's terms, seeing information presented slowly and inconveniently, with only the clumsiest control over its form or content. When the machine is required to play the game on the human's terms, presenting a pageful of attractively (or even legibly) formatted text, graphs, or pictures in the fraction of a second in which the human can pick out a significant pattern, it is the other way around: people are fast, and machines are slow. Indeed, it is beyond the current state of the art for a machine to keep up with a person. But the Alto system is a step in this direction.

So a personal system should present and accept information in a form convenient for a person. People are accustomed to dealing with ink on paper; the computer should simulate this as well as possible. It should also produce and accept voice and other sounds. Of these requirements, we judged most important the ability for the machine to present images, and for its user to point at places in the image. The Alto can do this quite well for a single 8.5″ × 11″ sheet of paper with black ink; it puts no restrictions on the form of the images. It cannot read images, nor can it handle sound; other people's experience suggest that these abilities, while valuable, are much less important.

The Alto system is *distributed* because everything in the real world is distributed, unless it is quite small. Also, it is implicit in our goals that the computer is quintessentially a communication device. If it is also to be personal, then it must be part of a distributed system, exchanging information with other personal computers. Finally, many things need to be shared, not just information, but expensive physical objects like printers and tape drives.

Finally, the Alto system is a *computing* system, adaptable by programming to a wide and unpredictable variety of tasks. It is big and fast enough that fitting the programs into the machine is not too hard (though the Alto's address space and memory size limitations were its most serious deficiency). And programming is possible at every level from the microcode to the user of an application. It is interesting, though, that user programming plays a much smaller role in the Alto system than in Unix [23] or Emacs [46]. Users *interact* with the system, nearly to the exclusion of *programming* it. In retrospect it seems

that we adopted this style because interacting with the Alto was so new and enthralling, and that we went too far in excluding systematic programming facilities like the Unix shell or M-Lisp in Emacs. The same mistake was made in early time-sharing systems such as CTSS [8] and the SDS 940 [26], whose builders were enthralled by a cruder kind of interaction, so we should have known better.

1.2 Schemes

"Systems resemble the organizations that produce them." *Conway*

The Alto system was affected not only by the ideas its builders had about what kind of system to build, but also by their ideas about how to do computer systems research. In particular, we thought that it is important to predict the evolution of hardware technology, and start working with a new kind of system five to ten years before it becomes feasible as a commercial product. If the Alto had been an article of commerce in 1974, it would have cost $40,000 and would have had few buyers. In 1984 a personal computer with four times the memory and disk storage, and a display half the size, could be bought for about $3500. But there is still very little software for that computer that truly exploits the potential of personal distributed computing, and much of what does exist is derived from the Alto system. So we built the most capable system we could afford within our generous research budget, confident that it could be sold at an affordable price before we could figure out what to do with it. This policy was continued in the late 1970's, when we built the Dorado, a machine with about ten times the computing power of an Alto. Affordable Dorados are not yet available, but they surely will be by the late 1980's. Our insistence on working with tomorrow's hardware accounts for many of the differences between the Alto system and the early personal computers that were coming into existence at the same time.

Another important idea was that effective computer system design is much more likely if the designers use their own systems. This rule imposes some constraints on what can be built. For instance, we had little success with database research, because it was hard to find significant applications for databases in our laboratory. The System R project [17] shows that excellent work can be done by ignoring this rule, but we followed it fairly religiously.

A third principle was that there should not be a grand plan for the system. For the most part, the various parts of the Alto system were developed independently, albeit by people who worked together every day. There is no uniform design of user interfaces or of system-wide data abstractions such as structured documents or images. Most programs take over the whole machine, leaving no room for doing anything else at the same time, or indeed for communicating with any other program except through the file system. Several different language environments are available, and programs written in one language cannot call programs in another. This lack of integration is the result of two considerations. First, many of the Alto's capabilities were new, and we felt that unfettered experimentation was the best way to explore their uses. Second, the Alto has strictly limited memory and address space. Many applications are hard enough to squeeze into the whole machine, and could not have been written if they had to share the hardware resources. Nor is swapping a solution to the shortage of memory: the Alto's disk is slow, and even fast swapping is inconsistent with the goal of fast and highly predictable interaction between the user and the machine.

There was some effort to plan the development of the Alto system sufficiently to ensure that the basic facilities needed for daily work would be available. Also, there was a library of packages for performing basic functions like command line parsing. Documentation for the various programs and subroutine packages was maintained in a uniform way and collected into a manual of several hundred pages. But the main force for consistency in the Alto system was the informal interaction of its builders.

The outstanding exception to these observations is the Smalltalk system, which was built by a tightly-knit group which spent a lot of effort developing a consistent style, both for programming and for the user interface. Smalltalk also has a software-implemented virtual memory scheme which considerably relaxes the storage limitations of the Alto. The result is a far more coherent and well-integrated world than can be found in the rest of the Alto system, to the point that several of the Alto's successors modelled their user interfaces on Smalltalk. The price paid for this success is that many Smalltalk applications are too slow for daily use. Most Smalltalk users wrote their papers and read their mail using other Alto software.

There was much greater consistency and integration of components in later systems that evolved from the Alto, such as Star and Cedar. This was possible because their designers had a lot of experience to draw on and 4-16 times as much main storage in which to keep programs and data.

The Alto system was built by two groups at PARC: the Computer Science Laboratory (CSL), run by

Robert Taylor and Jerome Elkind, and the Learning Research Group (LRG), run by Alan Kay. LRG built Smalltalk, and CSL built the hardware and the rest of the system, often in collaboration with individuals from the other computer-related laboratory at PARC, the Systems Science Laboratory. George Pake, as manager first of PARC and then of all Xerox research, left the researchers free to pursue their ideas without interference from the rest of Xerox.

The Star system, a Xerox product based on the Alto [43], was built by the System Development Division (SDD), run by David Liddle. The Dorado, a second-generation perconal computer [27] and Cedar, a second-generation research system based on the Alto [53], were built in CSL.

Table 1 enumerates the major contributors to the Alto and Cedar systems. Table 2 is a chronology of these systems.

1.3 What was left out

A number of ideas that might have been incorporated into the Alto system did not find a place there. In some cases we made deliberate decisions not to pursue them, usually because we couldn't see how to implement them on the Alto within an interactive application. Classical computer graphics, with viewports, transformations, clipping, 3-D projection and display lists, falls in this category; one implementation of a computer graphics package was done, but it didn't get used because it took up too much of the machine. High-quality typesetting (as in TeX [24]) is another; doing this interactively is still an unsolved problem. Linking together a large collection of documents so that the cross-references can be quickly and easily followed, as Engelbart did in NLS [13] is a third. Most of these ideas were incorporated into Cedar.

We also did very little with tightly-linked distributed computing. The 150 Altos at PARC could have been used as a multi-processor, with several of them working on parts of the same problem. But except for the Worm programs described in section 3, this was never tried. And, as we have already seen, the Alto system gave little attention to high-level programming by users.

Other things were tried, but without much success. We built two file servers that supported transactions, one low-level data base system, and several applications that used data bases. In spite of this work, by 1984 databases still did not play an important role in the Alto system or its successors. This appears to be because the data in our applications is not sufficiently well-structured to exploit the capabilities of a database system. Most Unix users apparently have had similar experiences.

We also tried to apply ideas from AI to office systems, with work on natural language understanding and on expert systems. But ten years of work did not yield systems based on these ideas that were useful enough to make up for their large size and slow speed. A full-scale Lisp system was built for the Alto, but the machine proved too small for it in spite of much effort; this system was later successful on larger machines.

One important idea which would have been a natural application for the Alto was simply overlooked: spreadsheets. Probably this happened because except for the annual budget planning and keeping track of purchase orders, there were no applications for spreadsheets in the research laboratory.

1.4 Overview

The Alto system software was begun in the middle of 1973. In October 1976 the Alto User's Handbook was published [25]. It describes a system that includes an operating system, a display-oriented editor, three illustrators, high-quality printing facilities, and shared file storage and electronic mail provided by local network communication with a time-sharing machine. There were also two programming languages and their associated environments. This was a complete personal distributed computing system; it met nearly all of the computing and information-handling needs of CSL and LRG.

The rest of the paper tells the story of the Alto system from the bottom up, except for the hardware, which is described elsewhere [56]. Section 2 covers *programming*: operating systems, programming languages, and environments. Section 3 presents the basic *communication* facilities: internet protocols, remote procedure calls, file transfer, and remote terminals. Section 4 discusses the *servers* which provide shared resources: long-term storage, printing, naming and mail transport.

The last two sections deal with the parts of the system that interact with users, which after all is the purpose of the whole enterprise. Section 5 considers the *user interfaces*: how the screen is organized into windows, how images are made, and how user input is handled. Section 6 describes the major *applications*, programs the user invokes to do some job with results that are useful outside the computer system. In the Alto system these are text editors, illustrators, electronic mail, and computer-aided design programs.

The conclusion reflects on the future of personal distributed computing and the nature of computer systems research.

Throughout I describe the Alto system in some detail, especially the parts of it that have not been described elsewhere. I also sketch its evolution into the Dorado system and the Xerox 8000 products (usually called Star). The "Dorado system" actually runs on three machines: Dolphin, Dorado and Dandelion. These have different microengines, but the same virtual memory structure, and all the major programming environments run on all three systems. They differ in performance and availability: the Dolphin, with about twice the power of the Alto and ten times the memory, first ran in 1978; the Dorado, ten times an Alto, in 1979; the Dandelion, three times an Alto, in 1980. Star runs on the Dandelion and was announced in 1981. A fourth member of the family, the Dicentra, was designed as a server and runs only the Mesa environment.

Readers may be surprised that so much of this paper deals with software which has little or nothing to do with the display, since 'screen flash' is the most obvious and most immediately attractive property of the Alto system. They should remember that the Alto system put equal emphasis on *personal* and on *distributed* computing, on *interacting* with the computer and on *communicating* through it. It also seems important to explain the programming environment that made it possible to develop the large quantity of software in the Alto system.

2 Programming environments

The Alto system is programmed in a variety of languages: BCPL, Mesa, and Smalltalk (see section 2.2). Each language has its own instruction set and its own operating system. The entire system has in common only the file system format on the disk and the network protocols. These were established by the BCPL operating system and did not change after 1976; they constitute the interface between the various programming environments.

2.1 Operating systems

The first Alto operating system [29], usually called the OS, is derived from Stoy and Strachey's OS6 [47]. It provides a disk file and directory system, a keyboard handler, a teletype simulator for the screen, a standard stream abstraction for input/output, a program loader, and a free storage allocator. There is no provision for multiple processes, virtual memory, or protection, although the first two were provided (several times, in fact) by software or microcode packages. The OS is written entirely in BCPL [40]. It was designed by Butler Lampson, and implemented by him, Gene McDaniel, Bob Sproull and David Boggs.

The distinctive features of the Alto OS are:

- its open design, which allows any part of the system to be replaced by a client program;
- the world-swap facility, which exchanges control between two programs, each of which uses essentially all of the machine;
- the file system, which can run the disk at full speed and uses distributed redundancy to obtain high reliability.

The OS is organized as a set of standard packages, one providing each of the functions mentioned (except for the stream abstraction, which is entirely a matter of programming convention). When it is initialized, all the packages are present and can be called by any program that is loaded. However, there is a *junta* procedure which can be used to remove any number of the pre-loaded packages and recover the space they occupy. Thus a program can take over nearly the whole machine. BCPL programs can include any subset of the standard packages to replace the ones removed by the junta. Alternatively, they can provide their own implementation of a function that has been removed by a junta, or do without it altogether. Thus the system is entirely *open*, offering services to its client programs but preempting none of the machine's resources. Other packages not normally loaded as part of the system, but available to be included in any user program, provide non-preemptive concurrent processes, internet datagrams and byte streams, program overlays, and other facilities that might have been part of the standard system.

At the base of the OS is a *world-swap* function which saves the entire state of the machine on a disk file and restores it from another file; this allows an arbitrary program to take control of the machine. The new program communicates with the old one only through the file system and a few bytes of state which are saved in memory across the world-swap. The world-swap takes about two seconds; it is used for bootstrapping, checkpointing, debugging (to switch between the program and the debugger), and switching back and forth between two major activities of a program.

The file system represents a file as a sequence of disk blocks linked by forward and backward pointers. By careful integration with the disk controller, it is able to transfer consecutive file blocks that are

contiguous on the disk at the full disk speed of 1 Mbit/second, while leaving time for non-trivial client computing; this performance allows world-swapping, program overlaying, and other sequential file transfers to be fast. In addition, each disk block contains the file identifier and block number in a *label* field of its header; this information is checked whenever the block is read or written. As a result, the disk addresses of file blocks can be treated as *hints* by the system, rather than being critical to its correct functioning and to the integrity of other files. If a disk address is wrong, the label check will detect the error, and various recovery mechanisms can be invoked. A Scavenger program, written by Jim Morris, takes about a minute to check or restore the consistency of an Alto file system; it can be routinely run by non-professional users. As a result of this conservative design, the file system has proved to be very reliable; loss of any information other than bits physically damaged on the disk is essentially unheard of. This reliability is achieved in spite of the fact that many programs besides the standard operating system have access to the disk.

The file system also runs on the 80 and 300 MByte disks that are interfaced to some Altos, and in this form is the basis of a successful file server (see section 4.2).

The Alto has a conventional hierarchical directory system which allows multiple versions of each file, with provisions for keeping a prespecified number of versions. The subdirectory and version facilities were added late, however, and did not enjoy widespread use. A directory is stored as an ordinary file which is processed by a variety of programs in addition to the file system.

A program called the Executive processes command lines and invokes other programs; it is much like the Unix Shell, but with far more primitive facilities for programming at the command level. From the point of view of the OS it is just another program, the one normally invoked when the machine is booted. From the point of view of a user it is the visible form of the OS.

Some of the design choices of the Alto OS would not be made in an operating system for a 1985 workstation; its much greater computing power and especially memory capacity make them unattractive. Separate address spaces simplify program development and concurrent execution of separate applications; virtual memory simlifies programming; a clean virtual machine makes it easier to port client programs to a new machine. But these comforts have a significant cost in machine resources: memory and response time. The open and unprotected character of the Alto OS were essential to the success of many Alto applications, both interactive programs like editors and illustrators, and realtime programs like the file, print, and gateway servers.

Distinct operating systems evolved for other programming environments. The Alto Lisp and Smalltalk operating systems were built using many of the Alto OS packages. Over time, these packages were replaced by new code native to the Lisp and Smalltalk environments. Mesa had its own operating system on the Alto from the start, but it was moddelled closely on the OS.

For the Xerox 8000 products, the Alto OS was replaced by a considerably bigger system called Pilot, which supports virtual memory, multiple processes, and a more elaborate file system [38]. Pilot was also used in the Cedar system, but was eventually supplanted by the Cedar nucleus, a much simpler system quite similar in spirit to the Alto OS. Cedar also has CFS, the Cedar File System [41], which provides a network-wide file system to Cedar programs by caching files from a server on the workstation. In CFS modified files are written back only by explicit request. This arrangement supports about 20 Dorados from a single Alto-based file server, using abont 30 MBytes of local disk on each Dorado for the cache. Almost any file server will do, since it need only support full file transfers.

2.2 Languages and environments

One of the distinctive features of the Alto as a programmer's system is the wide variety of programming environments it supports: BCPL, Smalltalk, Mesa, and Lisp. Each of these has its own language, which gives the environment its name, and its own instruction set, implemented by its own microcode *emulator*; the BCPL instruction set is always present, and there is a RAM with room for a thousand microinstructions, enough for one other emulator. Each also has its own compiler, loader, runtime system, debugger and library of useful subroutine packages. The ensuing subsections describe these parts for each of the Alto environments.

Distinct environments communicate only by worldswap, and through the file system. This isolation is not a good thing in itself, but it allows each system to stretch the resources of the machine in a different way, and thus to support effectively a particular programming style which would be impractical on such a small machine if all of the systems had to share a common instruction set and operating system.

2.2.1 BCPL

BCPL [40] is an implementation language quite similar to C (indeed, it is C's immediate ancestor). BCPL has a fairly portable compiler which Jim Curry ported to the Data General Nova, and thence to the Alto. Curry also wrote a loader for compiled code. It can produce an executable image or an overlay file which can be loaded and unloaded during execution; overlays are used in several large programs to make up for the limited address space. The Swat debugger, built by Jim Morris, is the other component of the BCPL programming environment; it understands BCPL's symbols, but is basically a machine-language debugger. The entire BCPL environment is very much like the C environment in Unix. Most Alto software was programmed in BCPL until 1977. By 1978 new programs were being written almost entirely in the other languages.

2.2.2 Mesa

Mesa [15] is an implementation language descended from Pascal. Its distinguishing features are:

- Strong type-checking, which applies even across separate compilation units.
- Modules, with the interface to a module defined separately from its implementation. Inter-module type-checking is based on the interface definitions; an implementation can be changed without affecting any clients [30].
- Cheap facilities for concurrent programming, well integrated into the language [28].
- A very efficient implementation, which uses an instruction set highly tuned to the characteristics of the client programs [20]. Compiled Mesa programs are typically half the size of similar C programs for the VAX, for example. The instructions are called byte-codes; many are a single byte long, and none are longer that three bytes. The byte-codes are interpreted by a microcoded emulator.
- A symbolic debugger well integrated with the source language and the type system, which allows breakpoints to be placed by pointing at the source code, and values to be printed in a form based on their type.

Mesa was begun in 1971 on a time-sharing computer by Jim Mitchell, Chuck Geschke and Ed Satterthwaite, ported to the Alto in 1975 with the assistance of Rich Johnsson and John Wick, and adopted by SDD in 1976 as the programming language for all SDD products. It continued to evolve there into a complete integrated programming environment [49]. A separate path of evolution in CSL led to the Cedar system described below.

The main goal of the Mesa research and development was to support the construction of large software systems and to execute them efficiently. This was accomplished very successfully: by 1982 there were several systems programmed in Mesa that contain more than a quarter of a million lines of code each, and many more that are 20-50 thousand lines long.

2.2.3 Smalltalk

The Smalltalk system is an integrated programming environment for the Alto, the first one to be built. It consists of a programming language, a debugger, an object-oriented virtual memory, an editor, screen management, and user interface facilities [22]. The latter are discussed in section 5.

The Smalltalk language is based on *objects* and *classes*; each object has a class, which is a collection of procedures that operate on the object. A *sub-class* inherits the procedures of an existing class, and generally adds new ones. The class of an object acts as its type, which is determined at runtime; when a procedure is applied to an object, the name of the procedure is looked up in the object's class to find the proper code to execute. Smalltalk source code is compiled into byte-codes similar to the ones used for Mesa, although much more powerful (and hence slower). The bytecodes are quite close to the source, so the compiler is small and fast; a typical procedure can be compiled and installed in a few seconds on an Alto, without disturbing the rest of the running system.

Objects are allocated and garbage-collected automatically. A class is itself an object, as are the source and compiled code for each procedure, and the stack frame for an executing procedure. So everything in the system can be accessed as an object and manipulated by Smalltalk programs within the system [19]. The entire system contains 100-200 classes, ranging from streams and collections to processes and files, rectangles and splines [16].

The dynamic typing, garbage collection, accessibility of every part of the system, and ease with which a procedure can be added or changed without any loss of state make Smalltalk similar to Lisp as a programming environment. The object-oriented programming style and the subclass structure, along with the careful design of the hierarchy of standard classes, give

Smalltalk its unique character. The system is fairly easy to learn, and very easy to use for building prototypes. Its limitations are relatively slow execution and (in the Alto implementations) modest memory capacity; these factors have prevented production system from being built in Smalltalk.

There have been three generations of the Smalltalk language and system: Smalltalk 72, Smalltalk 76, and Smalltalk 80; the description above refers to the latter two. The first two run on the Alto, the last on the Dorado and also on several VAX, 68000, and 80286 implementations.

2.2.4 Lisp

Several researchers in CSL used PDP-10 Interlisp [54,55] for their programming in the 1970s. Interlisp became part of the Alto system through software which allows the Alto to be used as a very capable graphics terminal [45] which supports multiple fonts and windows, along with the standard Alto raster graphics primitives (see section 5). The resulting Interlisp-D system [52] was the first to integrate graphics into a Lisp system.

A complete Interlisp system was built for the Alto by Peter Deutsch and Willie-Sue Haugeland [9,10]. It uses many of the BCPL OS packages for operating system functions, and the Lisp runtime is also written in BCPL. Lisp programs are compiled into byte-codes much like those used for Mesa and Smalltalk; they fall about half-way between those in the amount of work done by a single instruction. As with Mesa, programs are about twice as compact as when compiled for a conventional instruction set. The system uses an encoded form for list cells, which allows a cell to be represented in 32 bits most of the time, even though addresses are 24 bits.

Alto Lisp worked, and was able to run most of PDP-10 Interlisp, but it was too slow to be useful, mainly because of insufficient memory. It therefore did not play a role in the Alto system. The implementation was moved to the Dorado, however, where after considerable tuning it has been very successful [7].

2.2.5 Cedar

With the advent of the Dorado in 1979, we saw an opportunity to improve substantially on all of our existing programming systems. After considerable reflection on the desirable properties of a programming environment for personal computing [11], CSL decided in 1979 to build a new environment, based on Mesa, to exploit the capabilities of the Dorado. The goals were significantly better tools for program development, and a collection of high-quality packages implementing the major abstractions needed to write applications.

Cedar added several things to the program development system: garbage-collected storage, dynamic types, complete program access to the representation of the running program, a version control system, and uniform access to local and remote files. It also had a large assortment of generally useful packages: an integrated editor for structured documents, a powerful graphics package, a relational database system, remote procedure calls, user-programmable interpretation of the keyboard and mouse, and a screen manager based on icons and non-overlapping windows [51,53].

By early 1982 the Cedar system was usable; by mid-1984 it had grown to about 400,000 lines of code, ranging from the compiler and runtime support to an electronic mail system based on the Cedar editor and database system. It was possible to implement real-time servers such as a voice data storage system and a transactional file system, using the full facilities of Cedar.

Cedar was quite successful in overcoming the major limitations of the Alto programming environment. It also succeeded in providing good implementations for a number of widely-used high-level abstractions. However, an additional goal was to equal Lisp and Smalltalk in supporting rapid program changes and late binding, and here it was less successful. In spite of this limitation, it probably represents the state of the art in programming environments for workstations in 1984.

3 Communication

In order not to lose the ability of time-sharing systems to support sharing of information and physical resources in moving to personal computing, it was necessary to develop communication facilities far beyond the state of the art in 1973. The Ethernet provided the hardware base for this development. A great deal of work also went into the software.

The first generation of Alto communication software, called Pup for PARC Universal Packet [4], is based on ideas developed for the Arpanet in the late 1960's [33], with changes to correct known problems. Pup was originally designed by Bob Metcalfe, and implemented by him together with David Boggs and Ed Taft.

The model for communication was structured in four levels:

Level 0: Transport—various mechanisms for transporting a datagram from one machine to another: Ethernet, Arpanet, leased telephone lines, etc.

Level 1: Internet datagrams

Level 2: Interprocess communication primitives—byte system protocol (BSP), rendezvous/termination protocol, routing table protocol, etc.

Level 3: Data structuring conventions—file transfer protocol (FTP), remote terminal protocol (Telnet), mail transfer protocol (MTP), etc.

Internet datagrams are the common coin, decoupling higher levels from the details of the packet transport mechanism. Unlike other systems based on local networks, such as the Cambridge ring [36] or the Apollo Domain system [31], the Alto system takes no advantage of the special properties of the Ethernet. Instead, all clients of the communication facilities use internet datagrams, which present the same interface whether the data travels over the Ethernet, over a telephone line, or through a dozen different transport mechanisms. Pup thus treats a transport mechanism just as the Arpanet treats an Imp-to-Imp telephone line; the role of the Imp is played by a *gateway* which routes each datagram onto a transport mechanism which is expected to take it closer to its destination.

A major difference from the Arpanet is that Pup offers no guarantee that a datagram accepted by the network will be delivered. When congestion occurs, it is relieved by discarding datagrams. This design reflects the expectation that most datagrams will travel over lightly loaded local networks; when links become congested, clients of the network may get disproportionately degraded service.

The main level 2 mechanism is the byte stream protocol, which provides a reliable full-duplex byte stream connection between two processes. It turns out to be fairly hard to use this kind of connection, since it is complicated to set up and take down, and has problems when unusual conditions occur. As in the Arpanet, however, this was of little concern, since the main customers for it are the level 3 file transfer and remote terminal services. These are normally provided by programs called FTP and Chat, invoked by users or their command scripts from the Executive command line processor [25]. FTP was written by David Boggs, Chat by Bob Sproull. There is also an FTP library package that programs can call directly to transfer files; this is used by the Laurel mail-reading program, and (in an abbreviated version) by many applications that send files to the printer.

Pup byte streams have fairly good performance. They are capable of transferring about .3 MBits/second between two Altos, using datagrams with 512 data bytes plus a few dozen bytes of overhead. This is an order of magnitude better than the performance exhibited by a typical byte stream implementation on a comparable machine. The difference is due to the simplicity of the Pup protocols and the efficient implementation.

Other important network services in the Alto system were booting a machine and collecting reports from diagnostic programs. These used internet datagrams directly rather than a level 2 protocol, as did one rather successful application, the Woodstock File System [50] described in section 4. But with these few exceptions, users and applications relied on FTP and Chat for communication services.

The only truly distributed program built in the Alto system other than the Pup internet router, is the Worm [42], which searches for idle machines into which it can replicate itself. It isn't entirely clear why more attempts were not made to take advantage of the 150 Altos or 50 Dorados available at PARC. Part of the reason is probably the difficulty of building distributed programs; this was not alleviated by RPC until 1982 (see below). The lack of a true network-wide file system before 1984 was probably another factor.

The Xerox 8000 network products use an outgrowth of Pup called Xerox Network System or XNS [60]. XNS adds one important level 3 protocol, the Courier remote procedure call protocol designed by Jim White. Courier defines a standard way of representing a procedure call, with its argument passing and result returning, between two machines. In other words, it includes conventions for identifying the procedure to be called, encoding various types of argument (integers, strings, arrays, records, etc.), and reporting successful completion or an exceptional result. Courier uses XNS byte streams to transport its data.

At about the same time, as part of the development of Cedar, Andrew Birrell and Bruce Nelson in CSL developed a remote procedure call mechanism that uses internet datagrams directly [2]. They also built a *stub generator* that automatically constructs the necessary program to convert a local procedure call into a remote one, and a location mechanism based on Grapevine (see section 4.3) to link a caller to a remote procedure. Like Courier, Cedar RPC uses a Mesa interface as the basic building block for defining remote procedures. Unlike Courier, Cedar RPC is fast: two Dorados using Cedar RPC can execute a null remote procedure call in about a millisecond.

Earlier work by Nelson indicates that further optimization can probably reduce this to about 200 microseconds; currently Cedar RPC has no clients that are limited by the performance.

The goal of Cedar RPC was to make it possible to build distributed programs without being a communications wizard. A number of successful weekend projects have demonstrated that the goal was reached. An unexpected byproduct was independent implementations of the same RPC mechanism in Lisp, Smalltalk, and C for the 8088; all these clients found it easier to implement the Cedar RPC than to design a more specialized mechanism for their own needs.

4 Servers

The main use of distributed computing in the Alto system is to provide various shared services to the personal computers. This is done by means of *servers*, machines often equipped with special input-output devices and programmed to supply a particular service such as printing or file storage. Although there is no reason why a single machine could not provide several services, in fact nearly all the servers are Altos, and have no room for more than one server program.

4.1 Printing

Printing is the most complex and most interesting service in the Alto system. A considerable amount of research had to be done to build practical printers that can make high-quality hard copies of the arbitrary images the Alto screen can display. The end product of this work at PARC was a printing system that stores many thousands of typeset pages, and can print them at about 40 pages per minute. The quality is fairly close to a xerographic copy of a professionally typeset version. All the documentation of the Alto system and all the technical papers, reports and memos written in the computer research laboratories were stored in the system and printed on demand.

There are several aspects of a printing service. First, there must be a printing *interface*, a way to describe the sequence of pages to be printed, by specifying the image desired on each page as well as the number of copies and other details. Second, there must be a *spooler* that accepts documents to be printed and queues them. Third, there must be an *imager* that converts the image descriptions into the raster of bits on the page, and sends the bits to the printing hardware at the right speed.

The spooler is fairly straightforward: nearly all the printers include a disk, accept files using a standard file transfer protocol, and save them in the OS file system on the disk for printing. Since imaging consumes the entire machine, it is necessary to interrupt printing to accept a new file, but file transfers are fast, so this is not a problem. The interfaces and the imagers are worth describing in more detail.

The main point of a printing interface is to decouple creators of documents from printers, so that they can develop independently. Four printing interfaces were developed for the Alto and Dorado systems, each providing more powerful image description and greater independence from the details of printer and fonts. The first, devised for the XGP printer by Peter Deutsch, is rather similar to the interfaces used to control plotters and dot-matrix printers nearly fifteen years later: it consists of the ASCII characters to be printed, interspersed with control character sequences to change the font, start a new line or a new page, justify a line, set the margins, draw a vector, etc. The graphics capabilities are quite limited, and the printer is responsible for some of the formatting (e.g., keeping text within margins).

This interface, with its lack of distinction between formatting a text document and printing, proved quite unsatisfactory; there was constant demand to extend the formatting capability. All the later interfaces take no responsibility for formatting, but require everything to be specified completely in the document. Thus line and page breaks, the position of each line on the page, and all other details are determined before the document is sent to the printer. This clear separation of responsibilities proved to be crucial in building both good printers and good document formatters.

The second printing interface was designed by Ron Rider for use with the Ears printer, the first raster printer capable of high-quality printing. The graphics capability of the Ears printer is limited, and as a consequence the Ears interface can only specify the printing of rectangles; usually these are horizontal or vertical lines. It does, however, allow and indeed require the document to include the bitmaps for all the fonts to be used in printing it. Thus font management is entirely decoupled from printing in Ears, and made the responsibility of the document creator. The great strength of this interface is the complete control it provides over the positioning of every character on the page, and the absence of restrictions on the size or form of characters; arbitrary bitmap images can be used to define characters. Font libraries were developed for logic symbols so that logic drawings could be printed, and other programs generate new fonts automatically for drawing curves.

After several years of experience with Ears, the advent of two new printers stimulated the development

Pren Edit

of a new, printer-independent interface called Press, designed by William Newman and Bob Sproull. Its main features are a systematic imaging model, arbitrary graphics, and a systematic scheme for naming fonts rather than including them in the document, since font management proved to be too hard for document creators. Thus the printer is responsible for storing fonts, and for drawing lines and spline curves specified in the document. The device-independence of Press makes it possible to display Press documents as well as print them.

Press served well for about six years. In 1980, however, the development of general-purpose raster printing products motivated yet another interface design, this time called Interpress; it was done by Bob Sproull and Butler Lampson, with assistance from John Warnock [44]. The biggest innovation is the introduction of a stack-based programming language, which contains commands to draw a line or curve, show a character, etc., as well as to manipulate the stack and call procedures. The document is a program in this language, which is executed to generate the image. This design provides great power for describing complex images, without complicating the interface. Interpress also has extensive provisions to improve device-independence, based on experience in implementing Press for a variety of printers. It handles textures and color systematically.

Imagers evolved roughly in parallel with interfaces. The task of generating the 10-25 million bits required to define a full-page raster at 300-500 bits/inch is not simple, especially when the printer can generate a page a second, as most of those in the Alto system can. Furthermore, the fonts pose a significant data-management problem, since a single type-face requires about 30 KBytes of bitmap, and every size and style is a different type-face. Thus one serif, one sans-serif and one fixed-pitch font, in roman, italic, bold, and bold italic, and in 6, 7, 8, 9, 10, 12, 14, 18 and 24 point sizes, results in 108 type-faces; actually the system uses many more.

The first imager, written by Peter Deutsch, drove the Xerox Graphics Printer. This machine is slow (5 pages/minute) and low resolution (200 dots/inch); it is also asynchronous, so the imager can take as long as it likes to generate each line of the raster. Only crude imaging software was developed for it, together with a few fonts. Raster fonts were unheard of at the time, except in 5x7 and 7x9 resolution for terminals, or at very high resolution for expensive photo-typesetters. The XGP fonts were developed entirely manually, using an editor that allows the operator to turn individual dots in the roughly 20x20 matrix on and off. They had to be new designs, since it is impractical to faithfully copy a printer's font at such low resolution. These XGP fonts were later widely used in universities.

The second imager drives Ears, a 500 dot/inch, 1 page/second xerographic printer based on a Xerox 3600 copier engine and a raster output scanner developed at PARC by Gary Starkweather. Ears is controlled by an Alto, but there is a substantial amount of special-purpose hardware, about three times the size of the Alto itself, to store font bitmaps and generate a 25 MHz video signal to control the printer. Both the imager and the hardware were developed by Ron Rider, with design assistance from Butler Lampson. This machine revolutionized our attitude to printing. The image quality is as good as a xerographic copy of a book, and it can print 80 pages an hour for each member of a 40-person laboratory (of course we never used more than a fraction of its capacity). A second copy of Ears served as the prototype for the Xerox 9700 computer printer, a very successful product. Fonts for Ears were produced using the Fred spline font editor described in section 6.

As Alto systems spread, there was demand for a cheaper and more easily reproduced printing server. This required a new printing engine, new imaging hardware, and a new imager. The engine was the Dover, also based on the 3600 copier, but at 384 dots/inch; its development was managed by John Ellenby. Bob Sproull and Severo Ornstein developed the Orbit imaging hardware based on a design by Butler Lampson; this is about half the size of the Alto that drove it, a better balance. Sproull and Dan Swinehart wrote the Spruce imager to accept Press files and drive the Dover and Orbit. About 50 copies of this system were made and distributed widely within Xerox and elsewhere. It was cheap enough that every group could have its own printer. Spruce includes the font management required by Press, and is normally configured with an 80 MByte disk for font storage and spooling. Although Press can specify arbitrary graphics, Spruce handles only lines, and it can be overloaded by a page with too many lines or small characters. These limitations are the result of the fact that Orbit has no full-page image buffer; hence Spruce must keep up with the printer, and may be unable do to so if the image is too complex.

Another Press imager, confusingly also called Press, was built by Bob Sproull and Patrick Baudelaire to drive two slower printing engines, both at 384 dots/inch, and one able to print four colors. This imager constructs the entire 15 MBit raster on a disk, and then plays it out to the printer; this is possible because the engines are so much slower. It was the

first Press imager, and the only one able to handle arbitrary images, by virtue of its complete raster store. Among other things, Press produced some spectacular color halftones.

The first Interpress imager was developed by Bob Ayers of SDD for the 8044 printer, part of the Xerox 8000 network product family. Subsequently Interpress imagers have been built for several other engines.

4.2 Storage

Although not quite as exciting as printing, shared storage services are even more necessary to a community of computer users, since they enable members of the community to share information. No less than five file servers were developed for the Alto and Dorado systems.

For its first three years, the Alto system used CSL's mainframe [14], running the Tenex operating system [3], as its file server. A Pup version of the Arpanet file transfer program, written by Ed Taft, provides access to the Tenex file system. Both Alto and Tenex have both user and server ends of this program, so either machine can take the active role, and files can also be transferred between two Altos.

The second file server, called WFS and written by David Boggs [50], is near the opposite extreme of possible designs. It uses a connectionless single-datagram request-response protocol, transfers one page of 512 bytes at a time, and implements only files; directories are the responsibility of the client. There is a very simple locking mechanism. This extreme simplicity allowed WFS to be implemented in two months on a Data General Nova; it was later ported to the Alto. Of course, only clients that provided their own naming for data were possible, since WFS had none. These included an experimental office system called Woodstock, the Officetalk forms-handling system, Smalltalk, and a telephone directory system.

As the Alto system grew, the file service load on Tenex became too great. Since the Alto has a disk controller for 300 MByte disks, it is possible to configure an Alto with several GBytes of storage. David Boggs and Ed Taft assembled existing Alto packages for files (from the OS), file transfer (from FTP), and directories (a B-Tree package written by Ed McCreight) to form the Interim File System (IFS). A considerable amount of tuning was needed to fit the whole thing into a 128 KByte Alto memory, and a new Scavenger had to be written to handle large disks, as well as an incremental backup system. IFS served as the main file server throughout the Alto system for at least seven years and dozens of IFS servers were installed.

IFS supports multiple versions of files, write locking, and a clumsy form of sub-directories. Over time it acquired single-packet protocols to report the existence of a file to support CFS (section 2.1), a WFS-style file server, an interum mail server, and access to Grapevine (section 4.3) for access control.

Concurrent with the development of WFS and IFS was a research project to build a random-access multi-machine file server that supports transactions and fine-grained locking. This became the Juniper system, designed and developed by Howard Sturgis, Jim Mitchell, Jim Morris, Jay Israel, and others [34]. It was the first server written in Mesa, the first to use an RPC-like protocol (section 3), and the first to attempt multi-machine computation. Juniper was put in service in 1977, but performance on the Alto was marginal. Later it was moved to the Dorado. The goals proved to be too ambitious for this initial design, however, and it was never widely used.

Cedar includes a second-generation file server supporting transactions. Called Alpine, it was built by Mark Brown, Ed Taft, and Karen Kolling [6]. Alpine uses Cedar RPC and garbage-collected storage; it also supports multi-machine transactions and fine-grained locks, using newer algorithms that benefit from the experience of Juniper and System R [17]. It is widely used in Cedar, especially to store databases.

4.3 Naming and mail transport

The Alto system initially relied on Pup host names to locate machines, and on the Tenex mail system for electronic mail. These worked well until the system grew to hundreds of machines, when it began to break down. It was replaced by Grapevine, designed and implemented by Andrew Birrell, Roy Levin, Roger Needham and Mike Schroeder. Grapevine provides naming and mail transport services in a highly available way [1]. It also handles distribution lists and access control lists, both essential for a widespread community.

Grapevine was the second Mesa server. It was put into service in 1980 and has been highly successful; dozens of servers support a community of about two thousand machines and seven thousand registered users. All the machines in the Alto and Dorado systems use it for password checking and access control, as well as for resource location and mail transport. It is also used to register exporters of RPC interfaces such as Alpine file service. The data base is replicated; in other words, each item stored on several servers; as a result it is so reliable that it can be depended on by all the other components of the

distributed system.

5 User interfaces

Perhaps the most influential aspects of the Alto system have been its user interfaces. These of course depend critically on the fact that the screen can display a complex image, and the machine is fast enough to change the image rapidly in response to user actions. Exploiting these capabilities turned out to be a complex and subtle matter; more than ten years later there is still a lot to be learned. The ensuing description is organized around four problems, and the Alto system's various solutions to them:

- organizing the screen,
- handling user input,
- viewing a complex data structure, and
- integrating many applications.

A final sub-section describes the facilities for making images.

5.1 Windows

As soon as the limited display area of the screen is used to show more than one image, some scheme is needed for multiplexing it among competing demands. There are two basic methods:

- *switching*, or time-multiplexing—show one image at a time, and switch quickly among the images;
- *splitting*, or space-multiplexing—give each image some screen space of its own.

Various approaches were tried, usually combining the two methods. All of them organize the major images competing for space, typically text documents or pictures of some kind, into rectangular regions called *windows*. All allow the user some control over the position and size of the windows. The windows either *overlap* or they *tile* the screen, arranged in one or two columns and taking up the available space without overlapping. Thus overlapping is a combination of splitting and switching; when two windows overlap, some of the screen space is switched between them, since only the one on top can be seen at any instant. Where windows don't overlap, the screen is split. Tiling, by contrast, is a pure splitting scheme.

With overlapping, the size of one window can be chosen independently of the size or position of others, either by the system or by the user. With tiling, this is obviously not the case, since when one window grows, its neighbor must shrink. As a practical matter, this means that a tiled system does more of the screen layout automatically.

An overlapped system can have any number of windows on the screen and still show the topmost ones completely, perhaps at the cost of completely obscuring some others so that the user loses track of them. In a tiled system, as more windows appear, the average size must become smaller. In practice, four or five windows in each of one or two columns is the limit. Either system can handle more windows by providing a tiny representation, or *icon*, as an alternative to the full-sized window. This is another form of switching.

A minor image, usually a menu of some kind, either occupies a sub-region of a window (splitting), or is shown in a *pop-up* window that appears under the cursor (switching). The pop-up temporarily obscures whatever is underneath, but only for a short time, while a mouse button is down or while the user fills in a blank. Thus it has an entirely different feeling from a more static overlapped window.

Figures 1-3 are typical screen arrangements from three systems. Smalltalk (figure 1) uses overlapping windows without icons, and the position of a window is independent of its function (unless the user manually arranges the windows according to some rule). The small skinny window is a pop-up menu. Smalltalk was the first system to use overlapping windows and pop-up menus. The Bravo editor (figure 2) uses one column of tiled windows, with a control window at the top, a message window at the bottom, and a main window for each document being edited, which may be subdivided to look at different parts. Cedar (figure 3) uses two tiled columns and rows of icons at the bottom (which can be covered up). This window system is called Viewers; much of its design was derived from Star. The top line or two of a window is a menu. Cedar also allows the entire screen image, called a *desktop*, to be saved away and replaced by another one; this is switching on a large scale. Markup has a pop-up menu scheme like Smalltalk's, but considerably more elaborate (figure 4).

5.2 Input

User input comes from the keyboard, the mouse buttons, or the cursor position, which is controlled by the mouse. The crucial notion is *clicking* at an object on the screen, by putting the cursor near it and pushing a button. There are three mouse buttons, so three kinds of click are possible; sometimes multiple clicks within a short interval have different meanings; sometimes the meaning of a click is modified by shift

keys on the keyboard; sometimes moving the mouse with a button held down has a different meaning. An interface for novices will draw sparingly from this variety of mouse inputs, but one designed for experts may use all the variations.

If the screen object clicked is a *menu button*, this gives a *command* that makes something happen; otherwise the object is *selected*, in other words, designated as an operand for a command, or some point near the object is designated, such as an insertion point for text. The selection is made visible by underlining or reversing black and white; the insertion point is made visible as a blinking caret, I-beam, or whatever.

Many interfaces also have *scroll bars*, thin rectangles along one side of a window; clicking here scrolls the document being viewed in the window so that another part of it can be viewed. Often a portion of the scroll bar is highlighted to indicate the position and size of the region being viewed relative to the whole document. A *thumbing* option scrolls the view so that the highlight is where the mouse is; thus thumbing a quarter of the way down the bar scrolls a quarter of the way into the document.

Frequently commands are given by keystrokes as well as, or instead of menu clicking. Nearly always the commands work on the current selection, so the selection can be changed any number of times before a command is given without any effect on the state. An expert interface often has mouse actions which simultaneously make a selection and invoke a command, for instance to open an icon into a window by double-clicking on it, or to delete an object by selecting with a shift key depressed.

Most Alto system interfaces are *modeless*, or nearly so: any keystroke or mouse click has the same general meaning in any state, rather than radically different meanings in different states. For example, an editor is modeless if typing the "A" key always inserts an "A" at the current insertion point; an editor is not modeless if typing "A" sometimes does this, and sometimes appends a file. A modeless interface usually has postfix commands, in which the operands are specified by selections or by filling in blanks in a form before each command is given.

Many of the user input ideas described here were first tried in the Gypsy editor (see section 6); others originated in Smalltalk, yet others in the Markup or Sil drawing programs (also described in section 6). One of the main lessons learned from these and other experiments is the importance of subtle factors in the handling of user input. Apparently minor variations can make a system much easier or much more difficult to use. In Bravo and Cedar this observation led to schemes which allowed the expert to rearrange the interpretation of user input, attaching different meanings to the possible mouse actions and keystrokes, and redesigning the menus. This decoupling of input handling from the actions of an application has also been successful in EMACS [46].

5.3 Views

More subtle than windows, menus, and double-click selections, but more significant, is the variety of methods in the Alto system for presenting on the screen a meaningful view of an abstract structure, whether it is a formatted document, a logic drawing, the tree structure of classes in Smalltalk, a hierarchical file system or the records in a database. Such a view represents visually the structure and content of the data, or at least an important aspect of it. Equally important, it allows the user to designate part of the data and change it, by pointing at the view and giving commands. And these commands have immediate visual feedback, allowing the user to verify that the command had the intended effect and reinforcing a sense of direct contact with the data stored in the machine.

There are many ways to get this effect; all the methods that have been developed over the centuries for presenting information visually can be applied, along with others that depend on the interactiveness of the computer. A few examples must suffice here.

Perhaps the most familiar is the "what you see is what you get" editor. The formatted document appears on the screen, complete with fonts, subscripts, correct line breaks, and paragraph leading; page layout should appear as well, but this was too hard for the Alto. The formatted text can be selected, and the format as well as the content changed by commands which show their effect immediately. Figure 2 is an example from Bravo.

The Smalltalk browser shown in figure 1 is a visual representation of part of the tree of Smalltalk classes and procedures. The panels show successive levels of the tree from left to right; the highlighted node is the parent of all the nodes in the panel immediately to its right. Underneath the panels is the text content of the last highlighted node, which is a leaf of the tree. Again, sub-trees can be selected and manipulated, and the effect is displayed at once.

The list of message headers in figure 3 is another example. Each line shows essential properties of a message, but it also stands for the message; it can be selected, and the message can be deleted, moved or copied to another file, read, or answered. The same methods are used in Gypsy and Star to represent a

hierarchical file system, using the visual metaphor of files in a folder and folders in a file drawer. The icons on the screen in Star, Cedar, and more recent systems like the Apple Macintosh repeat the theme again.

The idea of views was first used in Sketchpad, which provides a graphical representation of an underlying structure of constraints [48], but it was obscured by the fact that Sketchpad was intended as a drawing system. In the Alto system it appeared first in Gypsy and Bravo (for formatted text and files), then in the Smalltalk browser, and later in nearly every user interface. It is still a challenge, however, to devise a good visual representation for an abstraction, and to make the pointing and editing operations natural and fast.

5.4 Integration

An integrated system has a consistent user interface, consistent data representations, and co-resident applications. What does this mean?

A consistent user interface is one in which, for example, there is a single 'delete' command which deletes the selected object, whether the object is a text string, a curve, a file, or a logic element. Every kind of object has a visual representation, and operations common to several objects are specified in the same way and have the same results on all the objects. Most computer systems are constructed by amalgamating several applications and lack any consistency between the applications.

A consistent data representation is one that allows two applications handling the same kind of data to accept each other's output. Thus a document editor can handle the drawing produced by an illustrater or a graph plotting program, and a compiler can read the output of a structured document editor, ignoring the structuring information that isn't relevant to the syntax of the programming language.

Two applications are co-resident if control can pass from one to the other quickly and smoothly, without loss of the program state or the display state. If a text document editor allows pictures in the document, can pass control to the picture editor, and the picture can be edited while it appears on the screen along with the text, then the text and picture editors are co-resident. In the same way a spreadsheet, a data base query system, or a filing system might be co-resident with editors or with each other.

Integration is difficult for several reasons. Designing consistent interfaces is difficult, both technically and organizationally. Consistent data representations require compromises between the different needs of different applications; most Alto programmers were uninterested in compromise. Co-residency requires common screen-handling facilities that meet the needs of both applications, and enough machine resources to keep the code and data for more than one program readily available.

In spite of these problems, integration was always a goal of many Alto system applications, but success was limited. Gypsy has excellent integration of text editing and filing. The window system of Smalltalk provides some integration; the later Star and Cedar systems continue this, and all three have some consistency in user interfaces. All three also have a common data representation for simple text, and make it easy to copy text from one application to another as pipes do in Unix. Star goes furthest in consistency of interfaces and representations, and also has co-residency of all its applications. A great deal of effort was devoted to the user interface design [43] to achieve these results. Unfortunately, it pays a considerable penalty in complexity and performance. In Cedar some applications, notably the Tioga structured document editor, can be easily called as subroutines from other applications.

5.5 Making images

The Alto user interface depends heavily on displaying images to the user. These are of two main kinds: text and pictures. The application programs that allow users to make different kinds of images are described in the next section. They all, however, use a few basic imaging facilities which are discussed here.

The basic primitive for making images is BitBlt designed by Dan Ingalls [18,37]. It operates on rectangular sub-regions of two *bitmaps* or two-dimensional arrays of pixels, setting each pixel of the target rectangle to some function of its old value and the value of the corresponding source pixel. Useful functions are

- constant black or white, which sets the target black or white;
- source, which copies the source to the target;
- merge, which adds black ink from the source to the target;
- xor, which reverses the target color where the source is black.

There is also provision for a *texture* source which is an infinite repetition of a 4 x 4 array of pixels; this provides various shades of gray, striping, etc. The Alto has a micro-coded implementation of BitBlt which is quite fast.

BitBlt is used for rearranging windows and for scrolling images in a window. In addition, a single BitBlt can draw an arbitrary rectangle, and in particular a horizontal or vertical line, or it can copy or merge the image of a character from a source image called a *font* which contains all the characters of a typeface. Arbitrary lines and curves are drawn by procedures that manipulate the target image directly, since the rectangles involved are too small for BitBlt to be useful. In many cases, however, the resulting images are saved away as templates, and BitBlt copies them to the screen image.

In the Alto system applications make images by using BitBlt directly, or by using procedures to paint a character string in a given font or to draw a line or spline curve. Curves, once computed, are stored in a chain encoding that uses four bits to specify the direction of the curve at each pixel. There is also a package that handles arrays of intensity samples, built by Bob Sproull, Patrick Baudelaire and Jay Israel; it is used to process scanned images. Cedar has a complete graphics package that handles transformations, clipping, and halftoned representations of sampled images within a uniform framework [59].

6 Applications

Most people who look at the Alto system see the applications which allow users to do many of the major tasks that people do with paper: preparing and reading documents with text and pictures, filing information, and handling electronic mail. This section describes the major Alto system applications that are not programming systems: editors, illustrators, and filing and mail systems.

6.1 Editors

The Bravo editor was probably the most widely used Alto application [25]. It was designed by Butler Lampson and Charles Simonyi, and implemented by Simonyi, Tom Malloy, and a number of others. Bravo's salient features are rapid screen updating, editing speed independent of the size of the document, what-you-see-is-what-you-get formatting (with italics, Greek letters and justified text displayed on the screen), and semi-automatic error recovery. Figure 2 shows a Bravo screen, illustrating the appearance of formatted documents. Later versions also have style sheets, which introduce a level of indirection between the document and its layout, so that the layout can be changed systematically by editing the style sheet; the document specifies "emphasis" and the style sheet maps it to "italic" or to "underlined" as desired.

Bravo is a rather large program, since it includes a software-implemented virtual memory for text and fonts; layout of lines and paragraphs in a variety of fonts with adjustable margins, tab stops, and leading; mapping from a point in the displayed text back to the document; incremental screen update; a screen display identical to the final printed copy; hard-copy generation for Ears and Press printers as well as daisy-wheel devices; and page layout. Later versions have a document directory, style sheets, screen display of laid-out pages, hyphenation, a remote terminal service, an abbreviation dictionary, form letter generation, and assorted other features.

Good performance in a small machine comes from the representation of edits to the text and of formatting information, and from the method for updating the screen. The text is stored as a table of *pieces*, each of which is a descriptor for a substring of an immutable string stored in a file. Initially the entire document is a single piece, pointing to the entire file from which it was read. After a word is replaced, there are three pieces: one for the text before the word, one for the new characters, which are written on a scratch file as they are typed, and one for the text after the word. Since binary search is used to access the piece array, the speed is logarithmic in the number of edits. This scheme was independently invented by Jay Moore [35].

Formatting information is represented as a property record of 32 bits for each character (font, bold, italic, offset, etc.); since consecutive characters usually have the same properties, they are stored in run-coded form in a table much like the piece table. Changes in formatting are represented by a sequence of *formatting operators* attached to each piece. Thus to find the font for a character it is necessary first to find its entry in the run-coded table, and then to apply all the formatting operators in its piece. But it takes only a few instructions per piece to make a 60,000 character document italic. When an editing session is complete, the document is written to a file in a clean form, so that the piece table doesn't keep growing.

To make screen updating fast, the screen image is maintained as a table of lines, each containing its bitmap and pointers to the characters used to compute it. This table is treated as a cache: when an edit is done, any entries which depend on characters changed by the edit are invalidated. Then the entire screen is recomputed, but the cache is first checked for each line, so that recomputation is done only if the line is actually different.

Bravo has a clumsy user interface. The Gypsy editor, built by Larry Tesler and Tim Mott, uses much of the Bravo implementation but refines the user interface greatly. Gypsy introduced a modeless user interface, editable screen display of bold and italics, and an integrated file directory that is modified by the same commands used for editing. All these ideas were later used in many parts of the Alto system. The parallel development of Bravo and Gypsy is an illustration of the many elements required in a good application. Both the refined implementation methods of Bravo and the refined user interface of Gypsy are important; each required several years of work by talented people.

The Star editor evolved from Bravo and Gypsy, with further refinement of the user interface [43] and the integration of graphics (see the next section). The Star interface design in turn had considerable influence on later versions of Bravo, from which Microsoft Word was then derived. It also influenced the Cedar document editor, Tioga, which is distinguished by its support of tree-structured documents (derived from Engelbart's system [13]), and by a fully programmable style-sheet facility which can do arbitrary computations to determine the format of a piece of text.

There are also simple text editors embedded in the Smalltalk and Mesa programming environments. These allow plain text files to be created, read, and modified. They are used for editing programs and for viewing *typescripts*, the output of programs that produce a sequence of unformatted lines.

6.2 Illustrators

The Alto system has a number of different illustrators for different kinds of pictures: pure bit-maps (Markup), spline curves (Draw), logic diagrams and other images involving only straight lines (Sil), and font characters (Fred). They all handle multi-font text as well, and all produce Press output for printing; most can show a screen image that exactly matches the printed image. Usually they work on only one picture at a time. None is integrated with a text editor; however, there is a separate batch program called PressEdit (written by William Newman), that can combine images from several Press files into a single file. PressEdit is used to assemble complete documents with text and figures; it has the drawback that the assembled document cannot be edited interactively. Several designs for integrated text and graphics editors stumbled over the limited size of the Alto.

William Newman's Markup was the first illustrator for the Alto [25]. It edits multi-page Press files, one page at a time, but it can only handle text and bitmap images, not lines or curves. Markup provides an assortment of brushes for painting into the bitmap, as well as operations for erasing and for moving or copying a rectangular region of the image. There are separate facilities for text strings, so that these can be printed at full resolution, since screen-resolution text looks terrible in hard-copy. Figure 4 is an example of the kind of picture that can be made. The Apple Macintosh's MacPaint is quite similar to Markup.

There are two very different illustrators that deal with synthetic graphics, in which the picture is derived from a set of mathematically defined curves. Draw, written by Patrick Baudelaire, builds up the picture from arbitrary lines and spline curves [25]. It has facilities for copying parts of the image and for applying arbitrary linear transformations. Curves can be of various widths, dashed, and fitted with various arrowheads. Thus very pretty results can be obtained; figure 5 shows some examples. The great variety of possible curves makes it difficult to use Draw for simple pictures, however, and it runs out of space fairly quickly, since it needs a full-screen bitmap (60 KBytes) and code for real arithmetic and spline curves, as well as the usual font-handling code, space for the fonts, the user interface etc. All of this must fit in the 128 KBytes of the Alto. Draw does not use the software virtual memory or overlaying techniques that allow Bravo to fit; these would have made the program quite a bit more complicated.

Sil was built by Chuck Thacker for making logic drawings [57]. Thus the main requirements were capacity for a complex drawing, a user interface well suited to experts who spend tens or hundreds of hours with the program, and a library facility for handling the images of logic components. Sil allows only characters and horizontal and vertical lines, which are easily handled by all the raster printers. There is a special font of bitmaps that represent and-gates, or-gates, resistors, and a few other specialized shapes, and a macro facility allows drawings of complex components to be built up from these. Since Sil drawings can be rather complex (see figure 6 for an example), redrawing the picture must be avoided. Instead, Sil draws a picture element with white ink to erase it, and in background redraws every element that intersects the bounding box of the erased element. This simple heuristic works very well. Somewhat to everyone's surprise, Sil is the illustrator of choice when curves are not absolutely required, because of its speed, capacity, and convenience.

For designing font characters with spline outlines Patrick Baudelaire built Fred. An Alto interfaced to

a television camera captures an image of the character, and the user then fits splines around it. The resulting shapes can be scan-converted at various resolutions to produce different sizes and orientations of the character for printing.

Sil and Fred are illustrators associated with batch-processing programs: a design-rule checker and wire-list generator for logic drawings made in Sil, and a battery of scan-conversion, font-editing and management software for Fred.

6.3 Filing

From its earliest days the Alto system has had the usual operating system facilities for managing stored information: directories can be listed and files renamed or deleted. This is quite clumsy, however, and many attempts have been made to provide a better interface. Most of these involve listing the contents of a directory as a text document. Lines of the listing can be selected and edited: deletion means deleting the file, changing the name renames the file, and so forth. Gypsy was the first system to use this scheme; it was followed by the Descriptive Directory System (DDS) built by Peter Deutsch [25], by Neptune built by Keith Knox, and later by Star's file folders. DDS and Neptune allow the user to control what is displayed by writing filters such as "memo *and not* report" which are applied to the file names. Star has a standard hierarchical directory system, which it manifests by allowing folders to appear in other folders.

Some other experiments with filing were more interesting. The Smalltalk browser is described in section 5.3 and illustrated in figure 1; it has been very successful. Cedar has a facility called Whiteboards, illustrated in figure 3, which deals with a collection of pages, each containing text and simple figures. A picture on one page can be a reference to another page; when the reference picture is clicked with the mouse, the page it points to is displayed. This is a convenient way to organize a modest amount of information. Finally, the electronic mail systems are heavily used for filing.

6.4 Electronic mail

The Alto system certainly did not introduce electronic mail, and in its early days the mail service was provided by Tenex. One of the most successful applications, however, was the Laurel user mail system built by Doug Brotz, Roy Levin, Mike Schroeder, and Ben Wegbreit [5]. Laurel provides facilities for reading, filing, and composing electronic messages; actual transport and delivery is handled by Grapevine or some other transport mechanism.

Laurel uses a fixed arrangement of three windows in a single column; see figure 7. The top window is a message directory, with one line for each message in the current folder. The other two windows hold the text of a message being read, and the text of a message being composed; the latter can be edited, and text can be copied from the middle window to the bottom. Above each window is a line of menu buttons relevant to that window; some of them have associated blanks which can be filled in with text when that button is clicked. A set of messages can be selected in the directory by clicking, and deleted, moved, or copied as a unit. Deleted messages are marked by being struck out in the directory; the actual deletion is only done when exiting Laurel, and can be undone until then.

Laurel can handle any number of message folders, and allows a message to be moved or copied from one folder to another. It also provides commands to forward or answer a message; these set up the composition window in a plausible way, but it is easy for the user to edit it appropriately. Message bodies are copied to the user's Alto from a mail server when the message is first retrieved. A message folder is represented as two Alto files, a text file for the messages and a table-of-contents file; there is a scavenger that reconstructs the latter from the former in case of trouble.

This apparently simple user interface is the result of about a man-year of design, and shows its value in the fact that most users never refer to the manual. This friendliness, together with the transparent view of the state provided by the message directory, and the high reliability of message retrieval and storage, made Laurel very popular. A variant of Laurel called Cholla is used as the backbone of a control system for an integrated circuit fabrication line; special messages called recipes contain instructions for the operator and the computer-controlled equipment, and messages are sent to the next station and to administrators and logging files as a job progresses through the system. The beauty of this system is that everything is immediately accessible to a person trying to determine the state or track down an error, and that the high reliability of the Grapevine transport mechanism makes lost data extremely unlikely.

Cedar has a mail facility called Walnut, patterned on Laurel, but using the standard Viewers windows and menus; it was built by Willie-Sue Haugeland and Jim Donahue. Walnut uses the Cedar entity-relationship database built by Rick Cattell to store messages; this turned out to be relatively unsuccessful, much to our surprise. It seems that a database

system does not have much to contribute to a mail system.

7 Conclusion

During the 1970's the Computer Science Laboratory and Learning Research Group at Xerox PARC built a complete personal distributed computing system: computers, display, local network, operating systems, programming environments, communication software, printing, servers, windows, user interfaces, raster graphics, editors, illustrators, mail systems, and a host of other elements. This Alto system did not have a detailed plan, but it was built in pursuit of a clear goal: to make computers better tools for people to think and communicate.

The Alto system has spawned a great variety of offspring. The Dorado system and the Xerox Star are the most direct line. Table 3 lists a number of commercial systems derived from the Alto. There have also been many in universities and research laboratories.

A dozen years of work have made it clear that these systems only begin to exploit the possibilities of personal distributed computing. Continuing progress in devices ensures that processors and networks will get faster, storage devices larger, displays clearer and more colorful, and prices lower. Compact discs, for example, allow half a gigabyte to be reproduced for a few dollars and to be on-line for a few hundred dollars; the impact will surely be profound.

These improvements and, even more important, a better understanding of the problems will lead to systems that are easier to program and use, provide access to far more information, present that information much more clearly, and compute effectively with it.

Acknowledgements

I am grateful to Bob Sproull, Peter Deutsch, David Boggs, Carol Peters and Cynthia Hibbard for comments on earlier versions of this paper, and to Ed McCreight and Adele Goldberg for supplying some of the figures.

CSL

Roger Bates	Dorado
Patrick Baudelaire	Draw, Fred, AIS, Puffin
Andrew Birrell	Grapevine
David Boggs	Alto OS, IFS, Scavenger, Dicentra
Jim Curry	BCPL
Peter Deutsch	XGP, Alto Lisp, DDS
Jerry Elkind	CSL manager
John Ellenby	Alto 2, Dover, Puffin
Chuck Geschke	Mesa
Jim Horning	Laurel, Cedar
Butler Lampson	Alto, Alto OS, Bravo, Mesa, Juniper, Dorado, Dandelion, Orbit, Interpress, Cedar
Roy Levin	Laurel, Grapevine, Wildflower, Cedar
Tom Malloy	Bravo
Ed McCreight	Alto, Exec, Dorado, design automation
Gene McDaniel	Alto OS, Dorado
Bob Metcalfe	Ethernet, Pup
Jim Mitchell	Mesa, Juniper
Jim Morris	Scavenger, Mesa, Juniper, Swat, Cedar
Severo Ornstein	Orbit, Dorado, Etherphone
Bill Paxton	Tioga
Paul Rovner	Cedar
Ed Satterthwaite	Mesa
Mike Schroeder	Laurel, Grapevine, Cedar
Charles Simonyi	Bravo, Juniper
Bob Sproull	Press, Alto OS, Chat, Orbit, Spruce, Interpress
Howard Sturgis	Juniper
Ed Taft	IFS, Pup, Dorado, Alpine
Bob Taylor	CSL Manager
Warren Teitelman	Lisp, Cedar
Chuck Thacker	Alto, mouse, Sil, design automation, Dorado, Dolphin
Ben Wegbreit	Laurel

SSL

Roger Bates	Mouse, Trident disk controller
David Boggs	Ethernet, IFS, Pup, FTP
Tim Mott	Gypsy, Officetalk
William Newman	Press, Markup, Officetalk
Bill Paxton	Officetalk, Tioga
Ron Rider	Ears
Larry Tesler	Gypsy, Smalltalk

LRG

Alan Kay	Smalltalk
Dan Ingalls	Smalltalk, Bitblt

Table 1: People and their projects

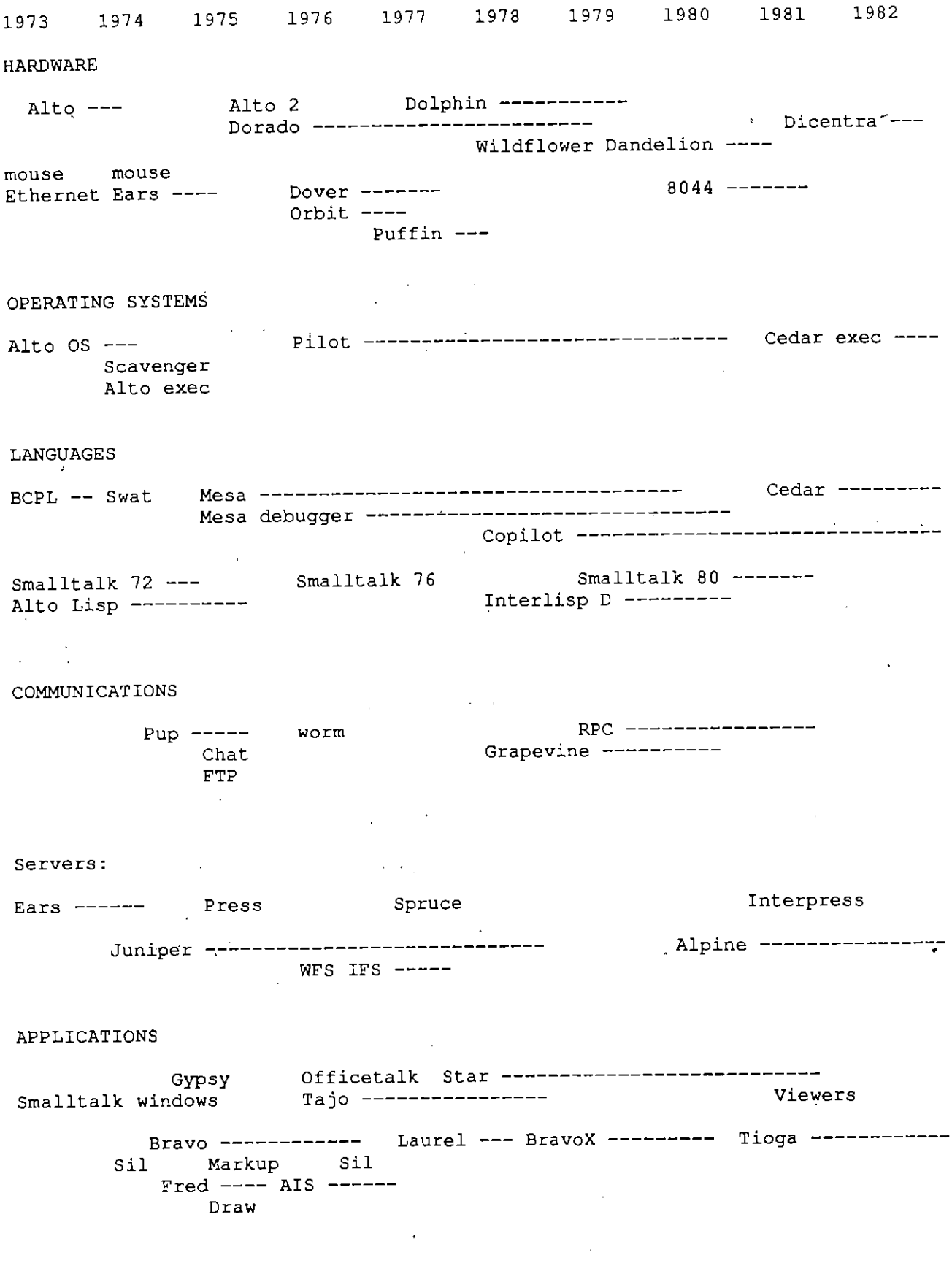


Table 2: Systems and chronology

Engineering workstations	Perq; Apollo; Sun; Tektronix; DEC; etc.
AI workstations:	MIT Lisp machine; Xerox 1100/1108/1132; Symbolic; LMI: etc.
Personal computers:	Apple Lisa, Macintosh; Xerox 8065.
Office workstations	Convergent NGen; Xerox 8010; Apple Macintosh; Grid; etc.
Graphics terminals	BBN Bitgraph; Bell Labs Blit; etc.
Local area networks	Ethernet/IEEE 802.3.
Networks protocols	ARPA IP/TCP; Xerox Network Services, Clearinghouse; etc.
Laser Printers	Xerox 9700, 5700, 8044; Imagen; Apple Laserwriter; etc.
Printing interfaces	Xerox Interpress; Adobe Postscript; etc.
Servers	3Com file server, Xerox 8044 and Apple Laserwriter print servers, etc.
User interfaces	Xerox 8010; Apple Macintosh; Microsoft Windows; etc.
Editors	Apple Macwrite; Microsoft Windows; etc.
Illustrations	Xerox 8010; Apple MacPaint, MacDraw; etc.

Table 3: Commercial systems descended from the Alto

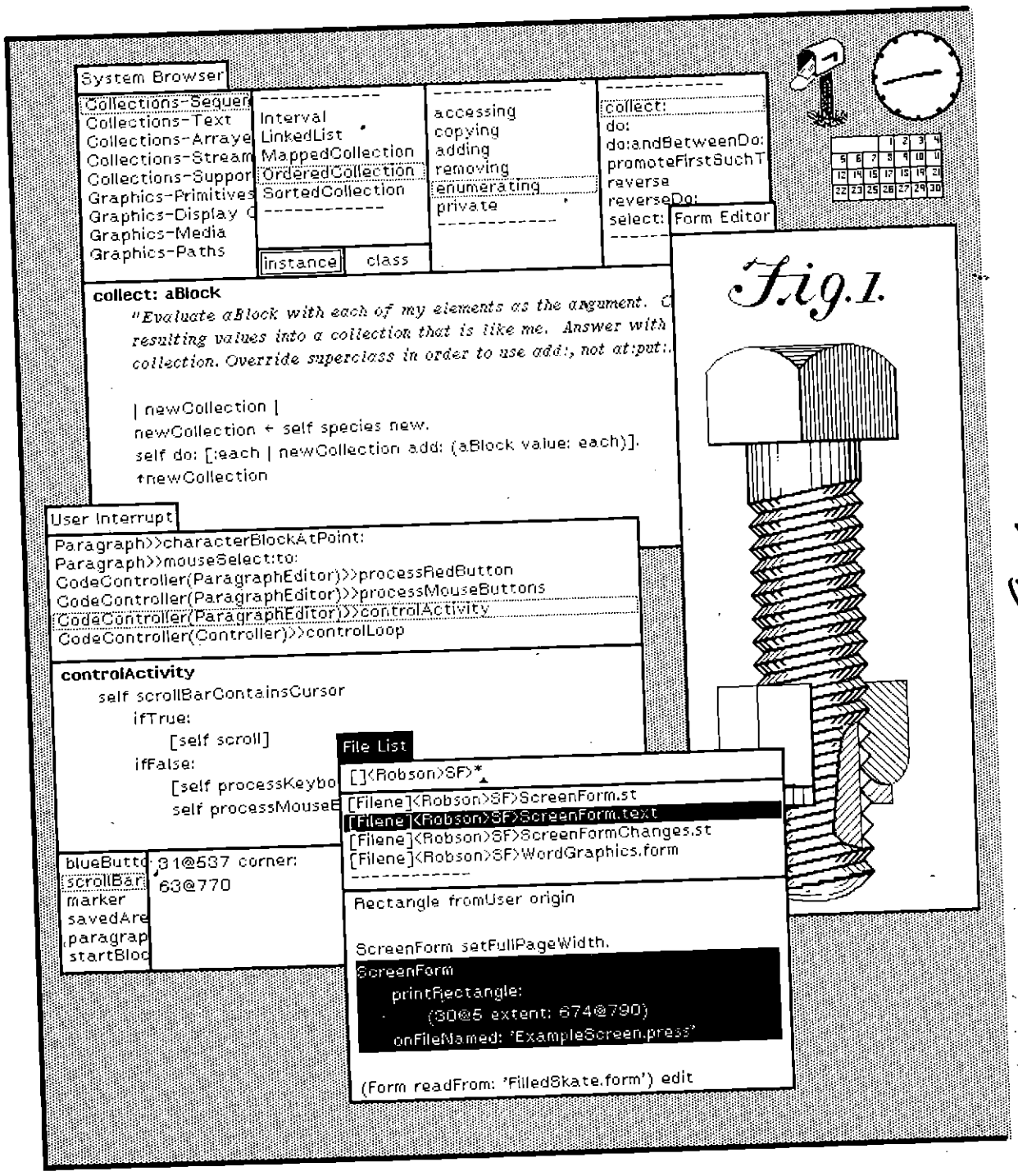
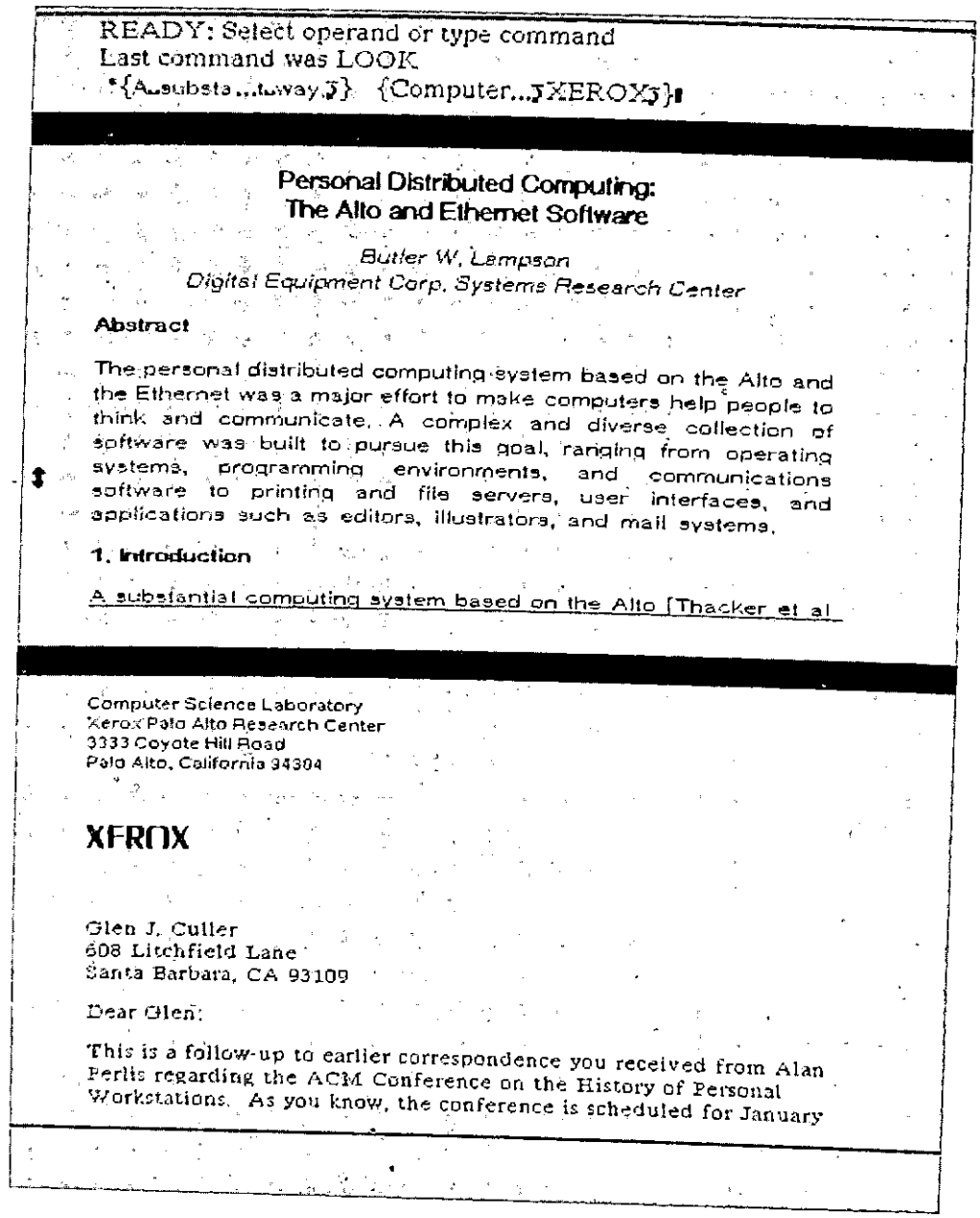


Figure 1: A Smalltalk screen; note the overlapping windows and the browser


READY: Select operand or type command
Last command was LOOK
*{A...subst...tway.5} {Computer...5XEROX5}

Personal Distributed Computing:
The Alto and Ethernet Software

Butler W. Lampson
Digital Equipment Corp. Systems Research Center

Abstract

The personal distributed computing system based on the Alto and the Ethernet was a major effort to make computers help people to think and communicate. A complex and diverse collection of software was built to pursue this goal, ranging from operating systems, programming environments, and communications software to printing and file servers, user interfaces, and applications such as editors, illustrators, and mail systems.

1. Introduction

A substantial computing system based on the Alto [Thacker et al

Computer Science Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

XEROX

Glen J. Culler
608 Litchfield Lane
Santa Barbara, CA 93109

Dear Glen:

This is a follow-up to earlier correspondence you received from Alan Perlis regarding the ACM Conference on the History of Personal Workstations. As you know, the conference is scheduled for January

Figure 2: A Bravo screen; note the formatted document

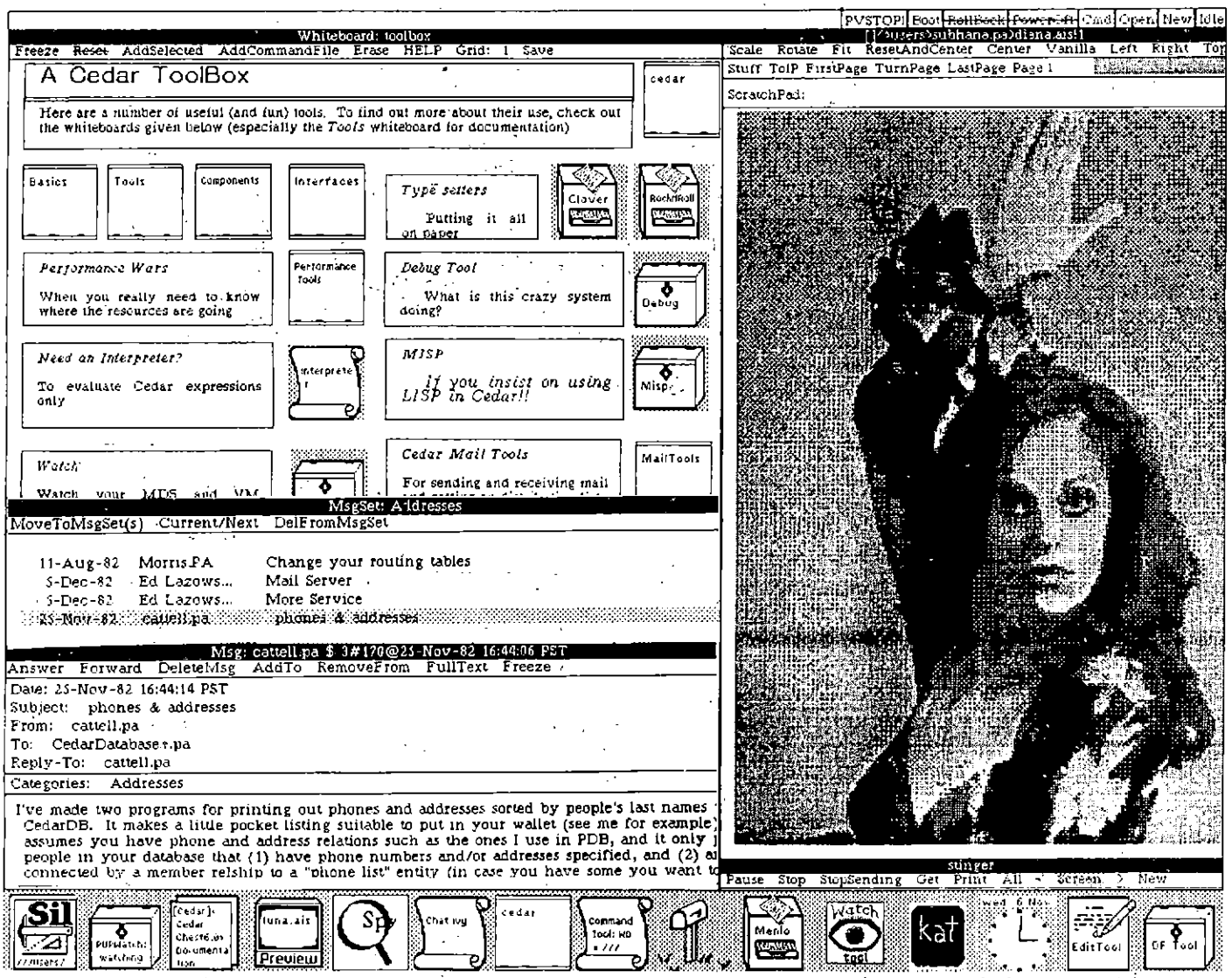


Figure 3: A Cedar screen; note the two-column tiling, the icons and the whiteboard

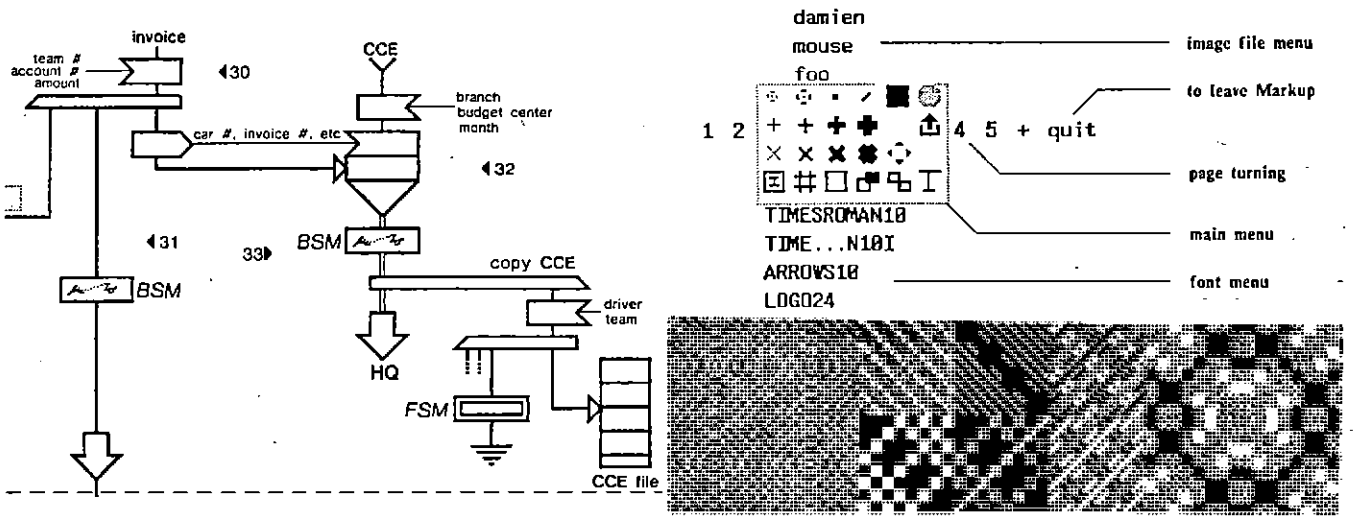


Figure 4: Typical Markup pictures

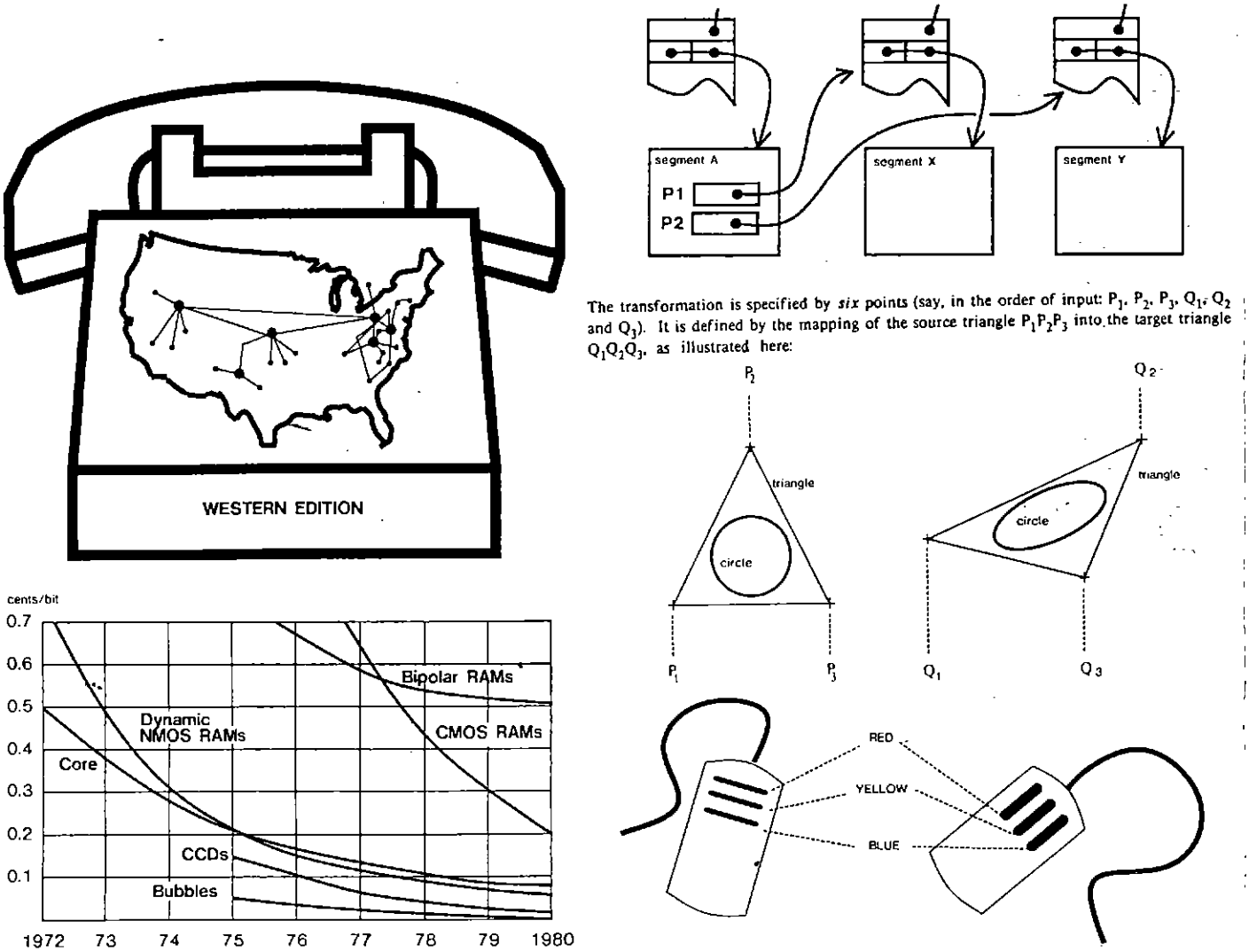


Figure 5: Typical Draw pictures

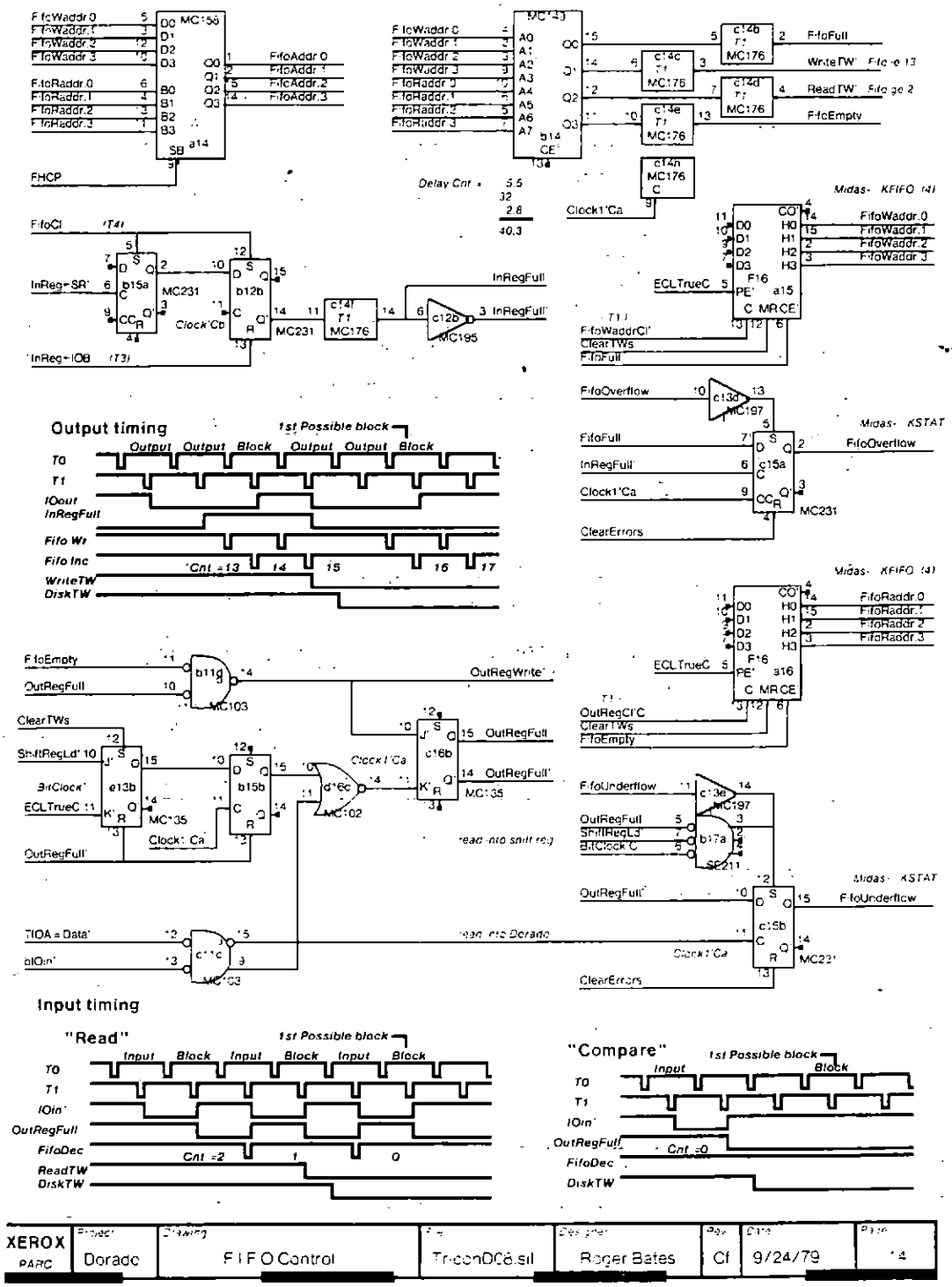


Figure 6: A logic drawing done in Sil

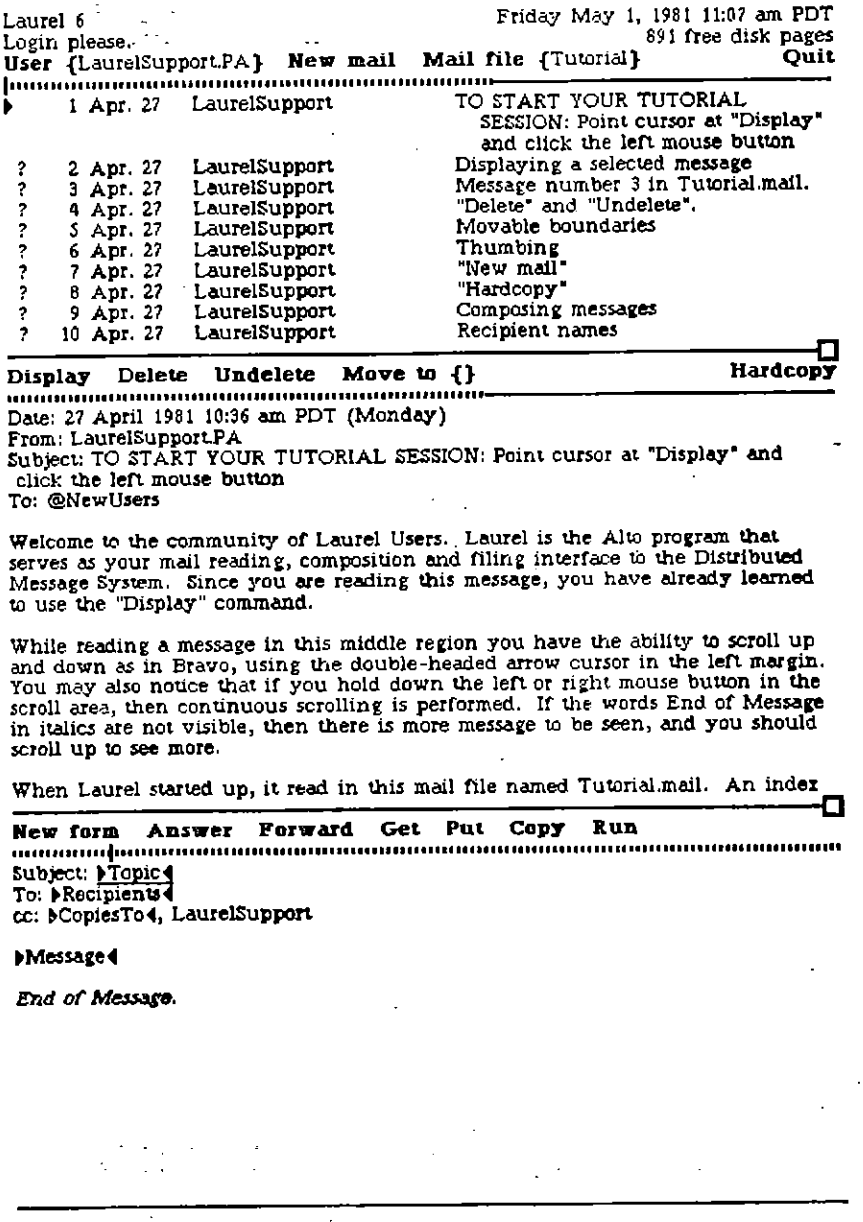


Figure 7: A Laurel screen

References

[1] A.D. Birrell et al. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[2] A.D. Birrell and B.J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[3] D.G. Bobrow et al. Tenex: A paged time-sharing system for the PDP-10. *Communications of the ACM*, 15(3):135–143, March 1972.

[4] D.R. Boggs et al. Pup: An internetwork architecture. *IEEE Transactions on Communications*, 28(4):612–624, April 1980.

[5] D.K. Brotz. *Laurel Manual.* Technical Report CSL-81-6, Xerox Palo Alto Research Center, 1981.

[6] M.R. Brown et al. The Alpine file system. *ACM Transactions on Computer Systems*, 3(2), November 1985.

[7] R.R. Burton et al. Interlisp-D: Overview and status. In *Proc. Lisp Conference*, Stanford, 1980.

[8] P.A. Crisman, editor. *The Compatible Time-Sharing System: A Programmer's Guide.* MIT Press, 2nd edition, 1965.

[9] L.P. Deutsch. Experience with a microprogrammed Interlisp system. *IEEE Transactions on Computers*, C-28(10), October 1979.

[10] L.P. Deutsch. A Lisp machine with very compact programs. In *Proc. 3rd International Joint Conference on Artificial Intelligence*, Stanford, 1973.

[11] L.P. Deutsch and E.A. Taft. *Requirements for an experimental programming environment.* Technical Report CSL-80-10, Xerox Palo Alto Research Center, June 1980.

[12] D.C. Engelbart. The augmented knowledge workshop. In *Proc. ACM Conference on the History of Personal Workstations*, January 1986.

[13] D.C. Engelbart and W.K English. A research center for augmenting human intellect. In *Proc. AFIPS Conference*, pages 395–410, 1968.

[14] E.R. Fiala. The MAXC systems. *IEEE Computer*, 11(5):57–67, May 1978.

[15] C.M. Geschke et al. Early experience with Mesa. *Communications of the ACM*, 20(8):540–553, August 1977.

[16] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[17] J. Gray et al. The recovery manager of the System R database manager. *ACM Computing Surveys*, 13(2):223–242, June 1981.

[18] D. Ingalls. The Smalltalk graphics kernel. *Byte*, 6(8):168–194, August 1981.

[19] D.H. Ingalls. The Smalltalk-76 programming system: Design and implementation. In *Proc. 5th ACM Symposium on Principles of Programming Languages*, pages 9–16, January 1978.

[20] R.K. Johnsson and J.D. Wick. An overview of the Mesa processor architecture. *ACM Sigplan Notices*, 17(4):20–29, April 1982.

[21] A.C. Kay. *The Reactive Engine.* PhD thesis, University of Utah, 1969.

[22] A.C. Kay and A. Goldberg. Personal dynamic media. *IEEE Computer*, 10(3), March 1977.

[23] B.W. Kernighan and R. Pike. *The Unix Programming Environment.* Prentice-Hall, 1983.

[24] D.E. Knuth. *TeX and Metafont: New Directions in Typesetting.* Digital Press and American Mathematical Society, 1979.

[25] B.W. Lampson, editor. *Alto User's Handbook.* Xerox Palo Alto Research Center, 1976.

[26] B.W. Lampson et al. A user machine in a time-sharing system. *Proc. IEEE*, 54(12):1744–1766, December 1966.

[27] B.W. Lampson and K.A. Pier. A processor for a high-performance personal computer. In *Proc. 7th Symposium on Computer Architecture*, pages 146–160, ACM Sigarch/IEEE, May 1980.

[28] B.W. Lampson and D.D Redell. Experience with processes and monitors in Mesa. *Communications of the ACM*, 23(2):105–117, February 1980.

[29] B.W. Lampson and R.F. Sproull. An open operating system for a single-user machine. *ACM Operating Systems Review*, 13(5), November 1979.

[30] H.C. Lauer and E.H. Satterthwaite. The impact of Mesa on system design. In *Proc. 4th International Conference on Software Engineering*, pages 174–182, September 1979.

[31] P.J. Leach et al. The architecture of an integrated local network. *IEEE Journal on Selected Areas of Communication*, SAC-1(5):842–856, November 1983.

[32] J. Licklider. Man-computer symbiosis. *IRE Trans. Human Factors in Electronics*, HFE-1:4–11, March 1960.

[33] J.M. McQuillan and D.C. Walden. The Arpanet design decisions. *Computer Networks*, 1(5):243–289, September 1977.

[34] J.G. Mitchell and J. Dion. A comparison of two network-based file servers. *Communications of the ACM*, 25(4):233–245, April 1982.

[35] J.S. Moore. *The TXDT Package - Interlisp Text Editing Primitives.* Technical Report CSL-81-2, Xerox Palo Alto Research Center, January 1981.

[36] R.M. Needham and A.J. Herbert. *The Cambridge Distributed Computing System.* Addison-Wesley, 1982.

[37] W.M. Newman and R.F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, 2nd edition, 1979.

[38] D.D. Redell et al. Pilot: An operating system for a personal computer. *Communications of the ACM*, 23(2):81–92, February 1980.

[39] H. Rheingold. *Tools for Thought.* Simon and Schuster, 1985.

[40] M. Richards. BCPL: A tool for compiler writing and system programming. In *Proc. AFIPS Conference*, pages 557–566, 1969.

[41] M:D. Schroeder et al. A caching file system for a programmer's workstation. *ACM Operating Systems Review*, 19(5), December 1985.

[42] J.F. Shoch and J.A. Hupp. Notes on the 'worm' programs–some early experiences with a distributed computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[43] D.C. Smith et al. The Star user interface: An overview. In *Proc. AFIPS Conf.*, pages 515–528, 1982.

[44] R.F. Sproull. *Introduction to Interpress.* Xerox Printing Systems Division, 1984.

[45] R.F. Sproull. Raster graphics for interactive programming environments. *Computer Graphics*, 3(3), July 1979.

[46] R.M. Stallman. EMACS: the extensible, customizable self-documenting display editor. In *ACM Sigplan Notices*, pages 147–156, June 1981.

[47] J.E. Stoy and C. Strachey. OS6–an experimental operating system for a small computer. *Computer Journal*, 15(2 and 3), May and August 1972.

[48] I. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. AFIPS Conf.*, pages 329–346, 1963.

[49] R.E. Sweet. The Mesa programming environment. *Sigplan Notices*, 20(7):216–229, July 1985.

[50] D. Swinehart et al. WFS: A simple shared file system for a distributed environment. *ACM Operating Systems Review*, 13(5), November 1979.

[51] D.C. Swinehart et al. The structure ofCedar. *Sigplan Notices*, 20(7):230–244, July 1985.

[52] W. Teitelman. A display-oriented programmer's assistant. In *Proc. 5th International Joint Conference on Artificial Intelligence*, pages 905–917, 1977.

[53] W. Teitelman. A tour through Cedar. *IEEE Software*, 1(4), April 1984.

[54] W. Teitelman et al. *Interlisp Reference Manual.* Technical Report, Xerox Palo Alto Research Center, 1978.

[55] W. Teitelman and L. Masinter. The Interlisp programming environment. *IEEE Computer*, 14(4):25–34, April 1981.

[56] C.P. Thacker. Personal distributed computing: The Alto and Ethernet hardware. In *ACM Conference on the History of Personal Workstations*, January 1986.

[57] C.P. Thacker. SIL—a simple illustrator for CAD. In S. Chang, editor, *Fundamentals Handbook of Electrical and Computer Engineering, Volume 3*, pages 477–489, Wiley, 1983.

[58] C.P. Thacker et al. Alto: A personal computer. In Siewiorek et al., editors, *Computer Structures:*

Principles and Examples, chapter 33, McGraw-Hill, 1982. Also CSL-79-11, Xerox Palo Alto Research Center (1979).

[59] J. Warnock and D.K. Wyatt. A device independent graphics imaging model for use with raster devices. *Computer Graphics*, 6(3), July 1982.

[60] J.E. White and Y.K. Dalal. Higher-level protocols enhance Ethernet. *Electronic Design*, 30(8):31–41, April 1982.

Butler et al
need to contact
Gordon Bell's
questions

---

ask some designers
from Xerox
influenced

Rob Pike
Steve Tibbs
Nick Wirth

THE LINC WAS EARLY AND SMALL

Wesley A. Clark

Clark, Rockoff, and Associates
One Sherman Square, New York, NY 10023

SUMMARY

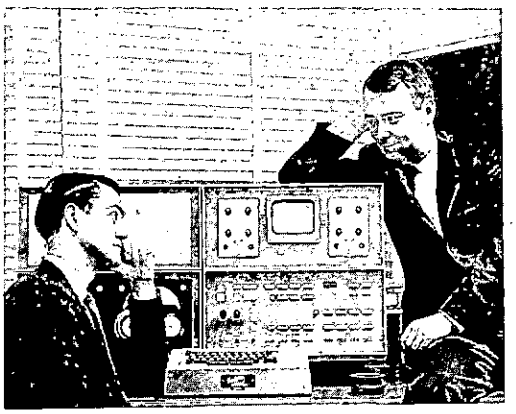
The LINC represents one of the earliest attempts to put the stored program computer into the form of a general instrument for laboratory use. In a deliberate departure from the technology of Timesharing then just beginning nearly two decades of development, the LINC was designed for use by individual experimenters and thus anticipated features of the modern personal computer and personal workstation. Built at M.I.T. in 1962, its immediate forebears were the TX-0, ARC-1, and L-1 computers, in turn direct descendents of the M.I.T. Whirlwind and MTC computers. Of course the LINC in its day was neither personal computer nor personal workstation but simply the LINC.

The LINC was an outgrowth of interactions between two M.I.T. groups of scientists and engineers: the Communications Biophysics Laboratory interested in the quantification of neuroelectric activity, and the Lincoln Laboratory Digital Computer Group engaged in the development of advanced computers. Twelve LINCs were placed initially in biomedical research laboratories across the country under a unique NIH/NASA-sponsored evaluation program. Ultimately more than 1200 LINC or LINC variants were manufactured commercially for worldwide use. The basic system design went on to influence the design of the DEC PDP-4 and PDP-5 computers, which in turn helped to pave the way to the PDP-8.

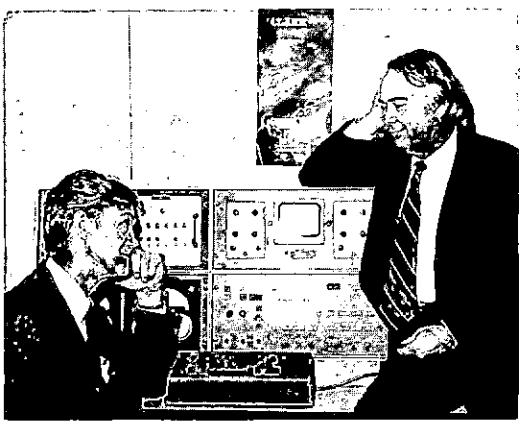
This paper is a revision and expansion of the paper "The Origins of the LINC", Proceedings of the IEEE Seventh Annual Conference of the Engineering in Medicine and Biology Society, ©IEEE 1985; additional figures and tables are taken from Computers in Biomedical Research Vol 2, R. W Stacy and B. D. Waxman (eds), ©Academic Press 1965. Permission of the publishers is gratefully acknowledged.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0133 $00.75



1963: The author and Mr. Charles E. Molnar with a LINC. The photograph was taken in Cambridge following the first phase of LINC Evaluation Program activities in which groups of scientists were given special training at a one-month summer course at M.I.T.



1981: The author and Dr. Charles E. Molnar with a LINC in a recreation of the above scene following a Digital Museum Lecture (see wall poster). Eighteen years grayed us both. Several of the original LINCs, also graying, were still in use in various parts of the world. (Photo courtesy of Digital Equipment Corporation)

INTRODUCTION

Twenty-two-plus years ago — in September, 1963 — the last of about twelve freshly assembled LINCs was safely delivered by moving van from M.I.T. to its new home in California. By itself, perhaps, this would not have been considered particularly noteworthy. But the event marked the successful completion of Phase I of a remarkable and unprecedented program. The twelve or so LINCs assembled during the very hot Cambridge summer of '63 had been put together by their new owners themselves. Each of these pioneers would take full responsibility for trial operation of the instrument as a workstation in his own biomedical research laboratory, and was expected to use it in the mode we have come to call, in Alan Kay's powerful paradigm, personal computing.

The participants in this innovative NIH/NASA-supported program had been selected on the basis of how suitable such a laboratory instrument would be in their ongoing or planned research activities. Each participant had agreed to assume individual responsibility for using and maintaining the machine throughout a subsequent eighteen-month period that defined Phase II of the program. The LINC Evaluation Program, as it was called, had been announced in *Science* and in descriptive brochures sent to academic and research institutions around the country. Candidates had been invited to submit proposals responsive to the scientific and technical objectives of the program. Computer programming or hardware experience was not required; the necessary fundamentals were to be learned in a one-month intensive training and kit assembly program to be conducted at M.I.T. by the LINC design team. Ultimate disposition of the assembled instruments was to be determined upon completion of Phase II.

More than seventy proposals had been received and studied by a specially created LINC Evaluation Board composed of distinguished scientists and engineers from research institutions across the country. After careful review the Board had finally accepted just twelve. These represented the fields of psychology, physiology, genetics, pharmacology, biophysics, and neurology. The selected participants had been divided into two groups to be convened in July and August respectively, with each principal investigator to be accompanied by one associate or colleague.

By mid-spring of 1963, all of the personal commitments had been made and much of the LINC hardware was in manufacture. But apart from one operating system/assembly program that had been developed and tested by simulation on a larger machine, the Lincoln Laboratory TX-2, the only LINC software in existence consisted of a few programs that had been written for an earlier demonstration prototype. With only two months to go, the final design of the LINC itself had not yet been completed.

How had all this come about? What was the LINC and what were its design objectives? Why did the field of biomedical research play such an important role in its development? I will try to recount something of the early history in answer to these questions. Like all such accounts of the introduction of a complex new instrument, the story is one of people, ideas, machines, and institutions and spans many years. As a principal instigator and designer of the LINC my perspective is limited to the events and circumstances that shaped only the earliest chapters. The full story must be left to others.

In what follows I will try to underscore and develop the title's assertion that the LINC, as personal workstations go, was early and small [1]. Early is evident; small, on the other hand, must be understood in the context of the memory capacity and the physical size of computers of the period. Since the machine's characteristics have been well documented (though not in the computer literature) I have indulged an urge to reminisce mostly about the people and events that really made the LINC experience such a rich and rewarding one. Some parts of the historical background may seem to be unnecessary embellishments, but I have included them in the hope that they will help in tracing the development of ideas and themes. The *dramatis personae* comprise extensive lists in both the computer and biomedical disciplines, and for the sake of simplicity I have kept professorial titles to a minimum and have arbitrarily 'Dr.'ed only those players who appear in character from the biomedical side of the stage. I hope that my friends will forgive me for this, as well as for any miscrediting of ideas and any errors or omissions of fact that may have settled into my own memory over the years.

BACKGROUND

Interacting with Whirlwind and MTC

Al Perlis and Doug Ross and I fondly remember the M.I.T. Digital Computer Laboratory Whirlwind computer, the machine on which I learned what little anyone knew about programming in 1952. One walked into it. What now sits comfortably on a small desktop, in those days required an entire room for the control consolery alone. Programming for its small electrostatic memory (1024 16-bit words on a good day) was a primitive affair carried out with the aid of heavily ruled coding forms on which to write out absolute-address instructions and octal numbers. [Confronted by the prospect of having to program in such a manner, Al immediately designed a new improved form that consisted of a large sheet of blank paper.] But its early users did indeed walk into the control room and for their assigned block of time, typically fifteen minutes or so depending on the time of day, the entire machine was theirs. In this regard the use of Whirlwind as a workstation of sorts was no different from that of many other early computers, especially in the universities. What makes the Whirlwind computer important in the history of the personal workstation is the fact that it was the first really high-speed machine with powerful interactive display capabilities [2].

But it was MTC that really drove the point home. MTC, the Memory Test Computer, was designed and built by Harlan Andersen and Ken Olsen to provide a working computer system in which to try out

the first ferrite core memory, a vacuum-tube driven 1K 16-bit unit engineered by Bill Papian then a graduate student under Jay Forrester [3]. Blocks of time measured in hours were available and the entire machine, quite similar in architecture to Whirlwind, now occupied only a single large room. Bristling with toggle-switches, pushbuttons, and indicator lights and provided with audio output as well as versatile CRT displays, it made interactive use a quite lively and memorable experience.

I suppose the idea of using even a very large computer such as the 1953 MTC just as one would use any laboratory tool was introduced to me by Belmont Farley, my teacher and collaborator in early computer simulation studies of neuron-like networks [4]. Belmont, a physicist by training and a man of considerable vision and strongly held convictions, joined the Digital Computer Laboratory shortly after my own move to M.I.T. to learn computerology. [I had almost become a physicist myself at one time, in what I now consider a close call; but that's another story.]

Belmont and I spent enormous numbers of hours interacting with MTC and with one another (he talked, I listened) as we tried out ideas, modified parameters, and studied displays of the simulated behavior of our strange little networks, grateful for such extensive access to such a powerful if not yet completely reliable machine. [A self-restarting test program was left running on MTC whenever the machine was otherwise unoccupied. In those pre-regulation days you could tune a radio to the frequencies being broadcast by MTC's long open-wire bus and hear the test program running, or trying to run, from several city blocks away. Sometimes when approaching by automobile for an extra-long simulation run in the dead of night the signal was clear enough for a decision to turn back home if things weren't going well.]

It was in these sessions that I began to learn from Belmont many of the basic attitudes toward computers that I hold firmly to this day: *Computers are tools; convenience of use is the most important single design factor — Big computers are for big jobs; small computers, for small jobs — Separate personal files are safer than files held in a common shared space — Digital computers should handle analog signals as well* — and so on.

In what would become a very important liaison, Belmont and I began to attend the seminars of the Communications Biophysics Laboratory, CBL, under Prof. Walter Rosenblith, later Provost of M.I.T. and presently Foreign Secretary of the National Academy of Sciences. Walter's group was concerned with the problem of quantifying neuroelectric phenomena. It was quite interdisciplinary in its approach and attracted students, post-doctoral fellows, and visiting scientists from many parts of the world. CBL had been developing various signal processing techniques. Much of CBL's work involved auto- and cross-correlation analyses based on data recorded on FM tape and then processed by analog devices. Work was just beginning on the analysis of the electrical signals from single neurons.

Our interaction with CBL started slowly. The Digital Computer Laboratory was absorbed into the recently established Lincoln Laboratory to become a component of Lincoln's massive program in air defense. Our base of operations shifted fifteen miles west to Lexington and we were assigned to a new group charged with advanced computer development together with Ken Olsen, Bill Papian, Dick Best, Jim Forgie, John Francovich, Ben Gurley, and many other members of the former Digital Computer Laboratory staff. Belmont and I somehow managed to keep up our pilot studies in network simulation with much travel back and forth to CBL and to the MTC in Cambridge.

Bill Papian and others, bolstered by the immediate success of the 1K MTC memory (shortly thereafter replaced by a 4K unit) now began to build the largest core-memory array thought feasible, a 64K 36-bit unit. This array would be thirty-six times the storage capacity of the 4K MTC array and would certainly constitute a big step in computer memory technology. In my new capacity as leader of a small sub-group in logic design I made sure that its potential for further simulation and modeling work was kept in mind.

The Lincoln TX-0 Computer

It was a time of transistors. Ken Olsen, in charge of the sub-group in advanced engineering development, joined me in proposing the construction of a vacuum-tube machine of considerable size, the TX-1, as a suitable vehicle for testing the large memory array [5]. When this idea was turned down by Laboratory management (wisely enough) Ken and others began to develop transistor circuits in earnest. I began the logic design of a new machine, the TX-2, which would have about the same architecture as that of the rejected machine but would take advantage of the new circuits. It is interesting to note that one of the main architectural features of the TX-1/TX-2 design — multiple program counters used in what I badly termed multi-sequence programming [6,7] — was later employed to great advantage by Chuck Thacker in the design of the Alto workstation [8].

I proposed that we first build a much smaller, primitively simple computer that I had designed and named the TX-0 [3,9]. [We had already used up "1"; later Ken and I would say that we didn't build the TX-1 because we didn't like its color scheme.] The 18-bit TX-0 would use only half of the large memory array and would be quite simple in logical structure, maximally RISCy, one might say today. When the second half of the array was completed the full array would then be installed in the TX-2 on which design and construction would have been proceeding. The TX-0 would then be retrofitted with a new memory unit of some kind and serve as a front end processor to the TX-2, or perhaps simply be dismantled. In the meantime it would have served in early evaluation of the new memory and transistor circuits and would have been more than adequate to the task of network simulation as well as to other tasks more directly relevant to the Lincoln effort. Not a bad plan for an R&D program. It was adopted. The TX-0 was built and a three-year effort to design and build the TX-2 was begun [10]. In addition to powerful CRT display units designed by Ben Gurley, the consoles of both machines would ultimately bristle in the best M.I.T. tradition.

The simulation work Belmont and I were carrying out had long had the interest and support of Bill Papian. Appointed Leader of the Advanced Development Group, Bill actively encouraged further strengthening of the interaction with the CBL group. Students and scientists from Rosenblith's laboratory began to try their hand at digital processing of neuroelectric data on the TX-0 and later on the TX-2. One of these students was Charles Molnar, who was to become a primary catalytic agent in promoting sound relationships within and between the two groups. Charlie, I soon learned, was extraordinarily competent and talented both as an electrical engineer and as a student of biophysics. He shared the conviction that computer tools would play an important part in what was beginning to be called "the biomedical sciences" and proceeded to master the operational and technical details of both the TX-0 and the unfinished TX-2. Charlie and I began to develop a basis of collaboration that would lead to the LINC and to many other developments over the years.

As the second half of the large memory array neared completion it became time to decide what to do with the TX-0. Bill Papian suggested replacing its memory with the group's new transistor-driven 4K memory unit (about the same capacity as that of the earlier vacuum-tube-driven MTC unit) and then moving the entire machine to the M.I.T. campus. This would give a wider group of students and faculty an opportunity to experience the kind of hands-on interactive computer use that characterized our work at Lincoln Laboratory in the tradition we were carrying forward from Whirlwind and MTC.

By that time both of the Cambridge machines, Whirlwind and MTC, had been completely committed to the air defense effort and were no longer available for general use. The only surviving computing system paradigm seen by M.I.T. students and faculty was that of a very large International Business Machine in a tightly sealed Computation Center: the computer not as *tool*, but as *demigod*. Although we were not happy about giving up the TX-0, it was clear that making this small part of Lincoln's advanced technology available to a larger M.I.T. community would be an important corrective step. The machine was then being used in some very exciting work in interactive programming by Jack Gilmore (unfortunately never published) that anticipated much of what we do today, but Bill's argument that the group's attention would be shifting to the TX-2 prevailed. Arrangements were made to replace the memory, enrich the instruction repertoire, and move the modified TX-0 to the campus in Cambridge.

But where exactly should it go? The logical place, it seemed to me, was Walter Rosenblith's laboratory. After all, CBL was then using the TX-0 computer more actively than any other group of visitors from the campus and expanded use by George Gerstein, Charlie, and other members of CBL was expected. To my surprise Walter declined to accept the responsibility. I therewith gained an important insight about what would work in a small laboratory setting and what would not. It seems in retrospect that the TX-0, small as it was for its time and demonstrably useful in CBL research, was still too much "the computer" and not sufficiently "an instrument". Its care and feeding and the constant need for justification might well have compromised CBL's research objectives. Walter's technical judgement, as usual, was correct.

The modified TX-0 eventually found a home under the larger umbrella of the Electrical Engineering Department, where it was used heavily by CBL and other groups and helped to train several generations of students in the interactive use of computers on-line. Somewhat later it was joined and largely upstaged by a PDP-1 computer (in some sense a production version of the modified TX-0) that had been donated to M.I.T. by the Digital Equipment Corporation and would later be used in some of the earliest experiments in Timesharing [3].

Timesharing had by then captured the imagination of much of the M.I.T. computer science community (not yet a science, of course; it would not be so until a few years later, when Al Newell, Al Perlis, and Herb Simon said, *Let there be Computer Science*, and there was). This seemed to me a bad idea and still does, the captive imagination aspect no less than the underlying premise itself. Improved access, yes; thrashing competition and waste; no. This is not to deny that Timesharing resulted in a huge and productive impetus to computer science at a critical time. Certainly it has given much gainful employment to computer scientists and manufacturers and salesmen over the past twenty years or so. But it seems to me that we could have done better than to divert so much of our attention and resources to trying to make good the promise of a patently unattainable "sensible simultaneity" for all. [I may not always be right, but at least I am consistent. When working out the multi-sequence programming idea some years earlier I had considered and rejected the notion of competitive time-sharing by independent human users as grossly inefficient; much later at the dawn of the Timesharing Era I had no qualms about turning down a suggestion of Ed Fredkin's to set up such a system for the multi-sequence TX-2 (much to his disappointment and, as I recall, disgust); in meetings of the M.I.T. Long Range Study Committee in 1961 I objected (to no effect) that campus-wide Timesharing so enthusiastically being rationalized as a panacea would not be able to deal with real-time work such as CBL's and moreover would inhibit the development of any interactive computing that involved complex displays. The M.I.T. Study Committee Chairman and I both declined to sign an otherwise unanimous final report. He thought it didn't promise nearly enough. Walter urged me to write a minority report, but I knew a steam roller when I saw one and went on to other matters.]

The ARC and the L-1

Not long after the decision to move TX-0 was made, Charlie and I met with Moise Goldstein and Robert Brown of CBL. The subject of discussion was the possibility of building a digital device to extract stimulus-evoked neuroelectric potentials from a background of unrelated electrical activity by means of a summation technique then being tried at CBL on analog equipment. Aha! A digital instrument for Walter! I immediately agreed to take on the design task.

The logic design of the instrument fell into place readily enough. I simply used many of the transistor logic plug-in modules already designed by Ken Olsen and Dick Best for the TX-2, and my colleague Henry Zieman neatly solved the problem of memory by working out the details of a very small 256-word core array. An 18-bit structure seemed about right for the job and nicely matched that of the TX-0 with which the instrument might later be in communication. I designed two simple operating modes to provide response averaging and amplitude histogram compilation, both modes to be wired into the control circuits. With electronics, control switches, lights, CRT, plotter, and paper-tape punch, the whole thing — even without an analog-to-digital converter — required a cabinet about the size of two refrigerators turned on their sides. Today we would think it a machine of rare ugliness (as an unkind journalist once said of my automobile). We named it the ARC, an acronym for Average Response Computer [11].

The ARC was completed in early 1958. Though not all that much smaller than the TX-0, at least it was portable and rolled through doorways. I rode it in triumph into the Communications Biophysics Laboratory and it was almost immediately put into service. After about a year of operating experience the CBL staff decided that a third mode for compilation of time histograms of single neuron activity would be useful, and this was duly wired into the ARC control. Over the next several years the ARC served in a wide variety of studies, teaching researchers many new aspects of the neuroelectric behavior of the brain. It also confirmed my belief that there were indeed useful things that small digital computers could do in the laboratory.

But wouldn't it have been better to program these operating modes rather than wire them into the control circuits? I had since taken just that approach in designing an extremely simple stored-program computer of very limited capability, the 256 10-bit word L-1, for use in a special project [12]. There was no doubt in my mind that the far greater flexibility of the stored-program approach was of enormous value even in computers small enough to be considered "instruments" for laboratory use.

At the first symposium of the Brain Research Institute held at UCLA in 1960, Belmont and I joined with Walter and several members of CBL and Dr. M. A. B. Brazier (of both CBL and the BRI) in discussing some of the viewpoints and accomplishments of the M.I.T. work [13]. In my presentation I contrasted the stored-program and wired-logic approaches:

> The TX-0 is a relatively small but powerful stored-program computer. A rough comparison of the ARC and the TX-0 computers is illuminating. Both machines operate on 18-bit binary numbers, control analog-to-digital converters, have cathode ray tube displays, can be controlled by the experimenter, and hold about the same number of circuits derived from the same body of electronic technology. The TX-0, however, has about 8000 registers of digital storage and is organized as a general-purpose device.
>
> It is quite simple to program the TX-0 to act very much like the ARC; in fact this has been done, requiring an investment of effort measured only in hours. The TX-0 can in addition generate quite varied displays of the data and of results of analysis, and can be programmed to carry out exceedingly lengthy and complex operations if desired. This flexibility of behavior is characteristic of the stored-program computer and is obviously of great value.
>
> It is extremely important, of course, to provide for procedural flexibility and easy access to machines of this type in order to realize their full potential. Ideally the researcher would have the general-purpose computer in his laboratory for use "on-line", enabling him to observe and act on the basis of the calculated results while the experiment is in progress.
>
> Stored-program computers like the TX-0 are beginning to appear in commercial form and there is reason to hope that these machines, or perhaps other general-purpose machines with a capability somewhere between that of the ARC and the TX-0, will find their way into the laboratory.

Well, there it was. Clearly, the thing to do was to go off and design a machine that would fill the niche I had just defined.

DESIGNING THE LINC

It has been both pleasurable and dismaying to look over my design notebooks after a lapse of nearly twenty-five years. I can easily relive the the exhilaration and sense of discovery in the pages detailing those rare ideas that "worked" and survived and were good. But there are other pages, too, some of them recording in distressing detail my false starts, muddled thinking, and irredeemably bad ideas. The design process is like that, of course; yet one always hopes that the overall record will reveal a higher order of thought and development. It is embarrassing to report, therefore, that the LINC design notebooks do not, on balance, document the workings of an orderly mind.

Following the October symposium at UCLA I increasingly began to sketch and doodle on various handy surfaces. It seemed to me that I now had most of the keys to the design process: a firm belief in the soundness of the goals and a good sense of the functional requirements, the general technology to be used, and the bounds of acceptable size, complexity, and cost. What I was trying to find was some gimmick, some architectural idea to start the process off in any promising direction. Taking the principal constraint to be the factor of acceptable cost I arbitrarily set a target of

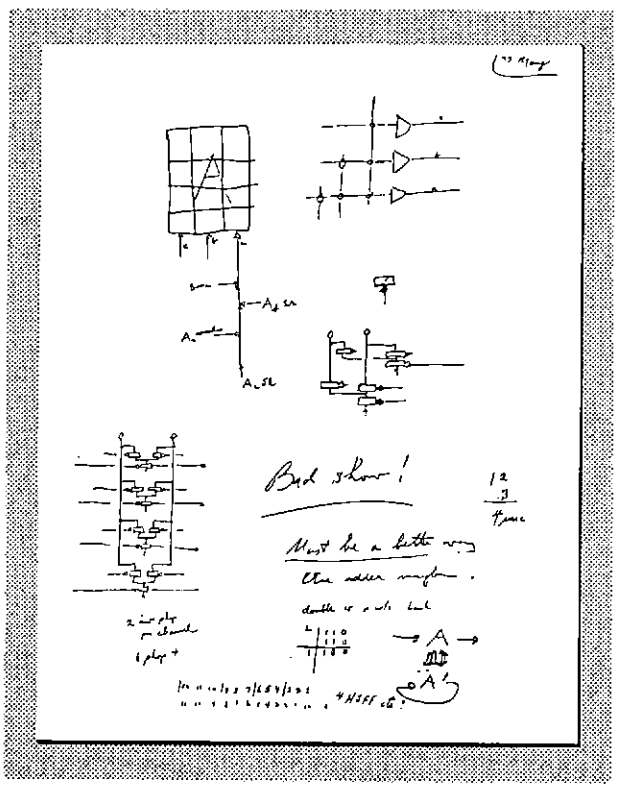
$25,000. This figure, relatively small for computers as we then knew them, seemed about right. It was about the size of one staff salary, for example, or about what a department head could authorize for equipment without the approval of higher management, and so on. Remarkably enough it was little more than twice the cost of the a-to-d converter used with the ARC. Was there some architectural scheme that might yield a useful laboratory-size computer subject to this cost constraint, something that would propel me out of the back-of-an-envelope phase and into the discipline of the engineering notebook?

First Attempts

Volume I of my three LINC notebooks carries the dates May 24 to July 4, 1961 — roughly Mother's Day to the Fourth of July. I had found a starting gimmick in the form of a serial-parallel scheme that appeared to reduce cost without seriously compromising speed. I would take advantage of a new family of plug-in circuit modules then being produced by the Digital Equipment Corporation. Ken Olsen and Harlan Andersen had left Lincoln Laboratory to found DEC in 1957 and were followed shortly thereafter by Dick Best, Ben Gurley (who later engineered the PDP-1), and others from the Lincoln group [3]. The circuit modules DEC was now manufacturing, forerunners of the integrated circuit chips that were not to appear for several more years, seemed quite suitable.

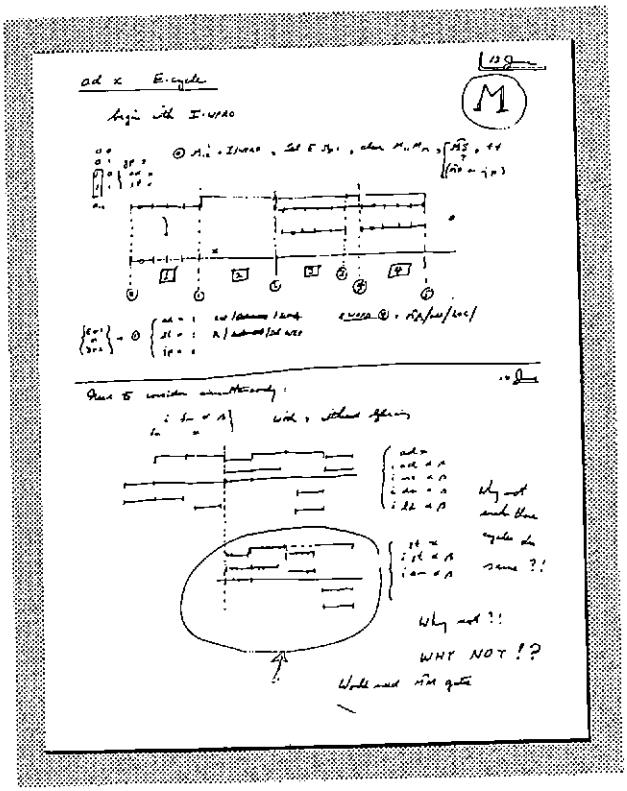
I knew that the machine would need two small tape units about the size of the "snapshot" unit I had proposed earlier for use with the TX-2 as a personal file input-output device and that my colleague Tom Stockebrand had tried to make. These units would have pocketably small reels for programs and data and would use block addressable magnetic tape like that of the TX-2's huge and frightening device for on-line files [14]. [The TX-2 tape unit used 14 inch reels of 3/4 inch tape, a very large inertial load. Its design was based on an observation of mine that the TX-2's multi-sequence capability made it possible to eliminate the usual requirement for finely controlled tape speeds, and I further proposed that the positions of all bit-cells on the tape be fixed by prerecording timing and blockmark tracks. In programming use, the resulting unit's tape speed varied markedly with wrap diameters, as did inertias and vibrational resonances. At top speed in block searching mode the tape's linear velocity reached 60 miles per hour! The whole room shook.]

Most of the work over these first two months went into thinking through the details of the serial-parallel architecture and an appropriate set of machine instructions. I knew that I had to build in a-to-d conversion channels and devote some of them to console potentiometers for convenient knob control of continuous variables (a good idea contributed by Belmont). I knew that CRT display would have to be inexpensive and would therefore use the same point-display technology that appeared in all of the M.I.T. machines. A more difficult task, however, would be thinking through the control for the tape units and developing the right set of block transfer modes for their use. The 1K or 2K 12-bit words of memory that seemed feasible under the cost constraint would not permit lengthy program routines for tape operation. These routines would therefore have to be wired into the central control circuits (of course there were no ROM chips in those early days). Block addressability of the tapes would help in running useful programs in the very small memory, the sort of thing we now do with disk systems.



The 27 May (1961) page from the LINC design notebooks. Early design was based on a serial-parallel logic using 4-bit shift register circuit modules made by DEC. Design progress was variably good.

I worked at home. Early in the design work I returned to the Laboratory briefly for an informal exposure of ideas to Charlie and Belmont and other members of the group, among whom were Severo Ornstein and a newcomer, Mary Allen Wilkes. There was a lively interest in what I presented as a "Linc". [I had thought that a generic name for this kind of computer would be useful and had decided on one that suggested the instrument's Lincoln Laboratory origins; Linc would be promoted to LINC by later events.] Following the meeting Severo and Mary Allen began to try programming with the proposed instruction set and I returned to the task of filling in design details. Charlie began to communicate his enthusiasm for the Linc scheme to CBL. Walter himself was later reported to be envisioning a relationship between a scientist and a computer similar to that between a scientist and a microscope. All was going well.



The 13 June (1961) page from the LINC design notebooks. Timing diagrams still appeared to support the serial-parallel scheme and new ideas were still emerging.

Alas, the serial-parallel approach turned out to be a false start. One month after its first exposure I had put enough of the design in place to establish the cost of the modules required by the central logic alone. Even without input-output devices or the core memory I had already used up my $25,000. Gad. The final entry in volume I is the single line: "Crisis in River City!"

The Prototype Linc

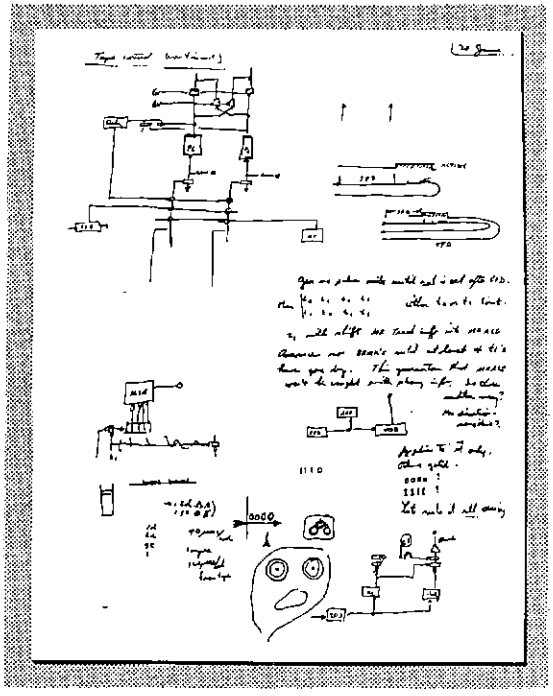
For the next several months I concentrated on simplifying the design and detailing the tape unit and its control, the most complex part of the machine and the largest single unknown. Much of my earlier work translated readily enough into a more straightforward 12-bit parallel form. Somewhat to my surprise this yielded a substantially lower cost. So much for gimmicks. Notebooks II and III record a great deal of work over the remainder of the year in refining the instruction set and developing simple operating modes for the tape units. Sketches for the tape transport mechanism and console appear here and there.

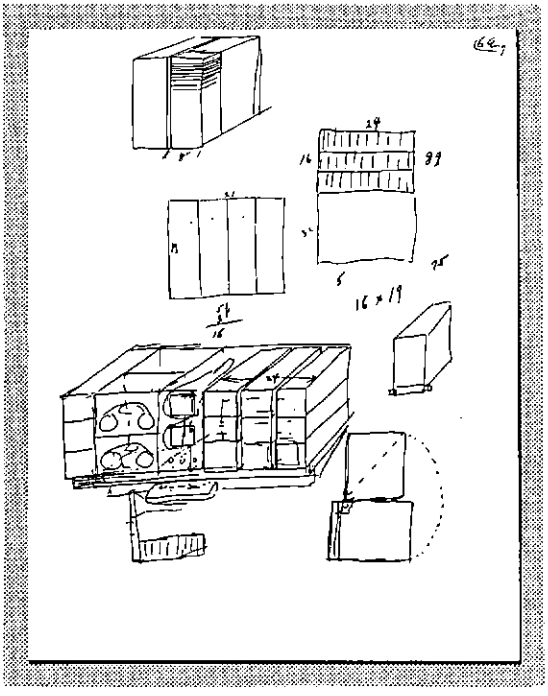
The need to build up a project effort at Lincoln began to demand more and more of my time as other members of the group joined in. Tom Stockebrand and I built a working model of the new tape unit, with Charlie contributing helpful insights from his experience with its fearsome progenitor, the giant tape unit of the TX-2. Severo designed and built a special subsystem to pre-mark tapes with the required fixed backbone of block addresses, timing signals, and control code patterns. Hershel Loomis and I assembled a tape unit exerciser/tester using DEC building block logic modules and established that the Linc tape equipment would indeed function as I had hoped. We were on the right track.

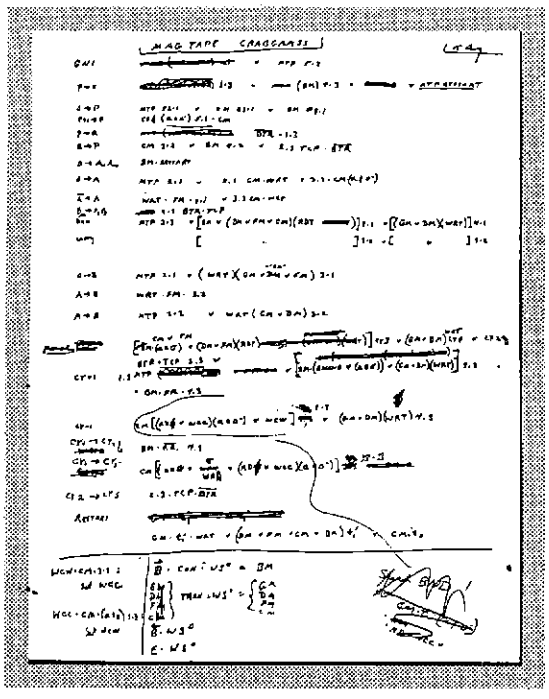
By early September the instruction timing diagrams had been completed in enough detail to enable Mary Allen to write a Linc system simulation program for the TX-2 and begin to develop a compact machine-code assembler for the proposed machine. Bill Simon, another newcomer and a remarkably creative scientist and engineer, designed a dual-scope alternative to the single-scope unit we eventually settled on, and began to write a number of test and demonstration programs. Norm Kinch, my right arm in matters of making things happen, began to produce large working drawings of system logic based on my notebook sketches and placed orders for parts as they emerged from the design. Severo and others helped with details of package layout and wiring lists. Charlie reviewed the design of the proposed 1K 12-bit memory system and watched over general engineering specifications. The final entry in Volume III was dated Christmas, 1961. Construction of the prototype Linc was already underway.

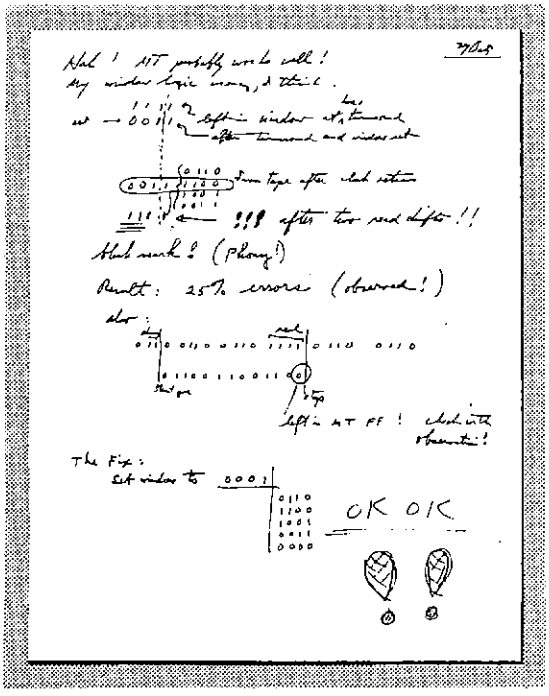
We hoped to complete the Linc prototype in time to take it to Washington for a demonstration in April at the forthcoming National Academy of Sciences Conference on Engineering and the Life Sciences. This had been urged by Walter and by Bill Papian as an excellent way to introduce the machine to a broad scientific and technical audience. Under this time schedule pressure but with tireless support from an enthusiastic technical staff, the Linc began to take shape. Hand-wiring of the electronics cabinet took its toll in anxiety and time over the ensuing weeks, as did the slow accumulation of special equipment and parts. But Norm kept the construction pretty much on schedule and the machine was completed in late February. It was working well enough for a Laboratory-wide demonstration at the end of March, 1962. The entire Linc group was justifiably proud of its accomplishment.

The demonstration prototype consisted of a set of four box-enclosed console modules, each connected by 20-ft long cables to a common electronics cabinet now the size of only *one* refrigerator; this general configuration would be used in all subsequent "academic" versions. One module, its box mostly empty, held a control panel that provided switches and register indicator lights (remember those?) together with speed and audio control knobs and so forth. A second module held a 5" CRT display adapted from a laboratory oscilloscope. A third module held the dual tape transport mechanics, while the fourth held a set of potentiometer knobs and jacks for analog input together with connectors for future input-output equipment. As an option for crowded laboratories, all modules could be removed from their boxes and mounted in standard equipment racks if desired. [Somewhat embarrassed



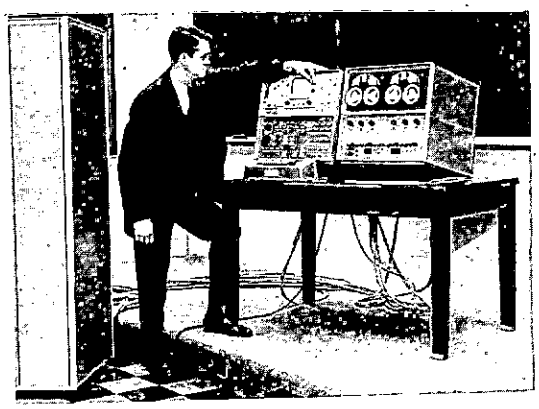






An abbreviated history of the LINC tape unit design. Upper left: June 28th sketches of parts of the tape turn-around control, including a picture of my foreboding of difficulties yet to come. Upper right: August 6th sketches of a possible arrangement of parts. Lower left: August 15th list of control logic expressions. Lower right: October 27th report of my realization that observed misbehavior of the prototype tape unit was caused by my design mistake. The unit worked well shortly thereafter. (From the 1961 LINC design notebooks).

by the size of the electronics cabinet, we adopted the point of view that it was only the console modules, taken together, that constituted the instrument itself; the rest was merely the electronics that made it all go and would be tucked out of sight in any convenient closet. We expected the cabinet eventually to disappear with advances in electronic packaging. It did, of course; but then so, eventually, did consoles.]



The Linc system demonstrated at the Lincoln Laboratory auditorium in March, 1962. Four console modules are connected by 20-foot cables to a cabinet holding the required digital electronics. The demonstration consisted of running a few simple programs toggled in by hand in advance.

The National Academy of Sciences Demonstration

Charlie, Norm, Bill Simon, and I took the Linc to Washington as planned and the National Academy demonstration was well if somewhat uncomprehendingly received [15]. Bill had arranged for closed circuit TV equipment and had set up several monitors about the meeting room. Charlie and I had spent the night before — *all* night — trying to fix some unexpected problem in the arithmetic element, crawling around the floor of our suite at the conference hotel with test equipment and soldering irons and discovering only by the dawn's early light that just outside our room there was a huge broadcast antenna tower that had been flooding everything with electromagnetic noise.

Norm commandeered one or two people from the hotel staff and moved the Linc to the meeting room for the presentation while Charlie and I dressed, Charlie in his Air Force uniform since at that time he was completing his military service at the Air Force Cambridge Research Laboratories and had to appear in offical garb. I stepped to the podium not knowing whether the machine was going to work or not, Charlie still frantically toggling in program parameters and trying out the demos. But at the last possible moment he slipped me a scrap of paper that said, "The following programs have my confidence: ...". The Linc performed perfectly. Whew! The only question asked, Charlie recalls, came from an older gentleman who seemed to be quite interested in the durability of the wiring insulation. He was from the Smithsonian.

NIH Interest

After the Academy meeting Norm moved the Linc to one of the laboratories of Dr. Robert Livingston, Scientific Director of both the Institute of Neurological Diseases and Blindness and the Institute of Mental Health. There Charlie connected the analog-to-digital input channels to a multiple electrode array implanted in the brain of one of the lab's mascots, a cat whose name, I believe, was Jasper. In short order he wrote a small program that displayed on the CRT the average neuroelectric responses of the behaving animal. The Linc was successfully performing its first scientific task.

The writer Sam Rosenfeld, in a background paper prepared for a seminar held at NIH in celebration of the twentieth anniversary of the LINC, quotes Dr. Livingston's recollection of the event [16]:

> It was such a triumph that we danced a jig right there around the equipment. No human being had ever been able to see what we had just witnessed. It was as if we had an opportunity to ski down a virgin snow field of a previously undiscovered mountain.

[Jasper merely purred and looked pleased.]



Jasper, one of several animals used in experimental work at a National Institutes of Health laboratory. The other end of the cable leading from the cat's head was connected to the Linc in a 1962 trial, the Linc's first scientific assignment.

It must be reported, however, that we did not have much success the next day when the Linc was moved to the laboratory of Dr. Mones Berman to see what it could do to isolate a very fast fluorescence transient produced in a particular biochemical reaction. Severo and Mary Allen answered an emergency call and came to Washington to help, but to no avail. Bill Simon would later write a superb program for the general problem of exponential decomposition but the limits of Linc speed had been reached.

Nonetheless, the interest of NIH had clearly been engaged. The National Academy of Sciences had already made the suggestion to Dr. James Shannon, Director of NIH, that he set up a committee to monitor and encourage the development of computer technology relevant to biomedical research. In response, Dr. Shannon had established the Advisory Committee on Computers in Research. This committee, among whose members were both Dr. Brazier and Bill Papian, had been following the M.I.T. activities closely. The Linc demonstrations at NIH had in fact been arranged by the committee's executive secretary, Dr. Bruce Waxman, who would shortly take further initiatives leading to the funding of a Linc evaluation program.

Linc Becomes LINC

The group at Lincoln had been joined by Dr. Thomas Sandel, a neurophysiological psychologist who had been conducting auditory research at CBL. Tom had organized and hosted an NSF-sponsored workshop the previous summer in which some of the M.I.T. computing techniques were shown to an invited group of biomedical researchers. He now proposed to use the Linc in a repeat of the workshop to be held in the summer of 1962. Plans were approved and a well-attended workshop was held during which several new Linc programs were written and new applications were investigated. [An amusing side benefit: In programming one of the data recording tasks we discovered that we had neglected to wire in a pathway for the signal specifying the initial direction of tape motion. Since the tape units were designed to search for designated blocks in either direction (now at a sedate though still uncontrolled speed of about 3 miles per hour), the occasional misbehavior had been self-correcting and had largely gone unnoticed.]

We began to think in terms of expanding the work. Bill Papian had been given to believe that under the right circumstances funds for support of a larger program in biomedical computing would be forthcoming from NIH. Charlie and Bill Simon and I were already beginning to modify and improve the design of the Linc so that a number of soundly engineered replicas could be made, and Tom Sandel was proposing to set up a small lab in space adjacent to the TX-2 for relevant biologically oriented work. But it was not to be. Lincoln Laboratory management, sensing that there would be serious organizational difficulties in administering such a program within its established framework, firmly rejected both the expansion and "wet" lab proposals. Instead, we were invited to find a more suitable home for any further work. In reporting to a stunned Linc design team the management's decision and my own decision to leave, I announced that "Linc" had just become "LINC", an acronym for Laboratory INstrument Computer.

The principals subsequently spent several disheartening months trying without success to find a "suitable home" for the continuation and extension of the work to which they were now deeply committed.

THE LINC EVALUATION PROGRAM

Toward the end of 1962, Walter Rosenblith put forth an imaginative proposal: M.I.T. would act as host institution in a multi-institutional, multidisciplinary endeavor to be called The Center for Computer Technology and Research in the Biomedical Sciences; faculty and staff from several New England universities would join in providing the scientific and technical substance of a long-range program; a Center Development Office would immediately be set up to work out details, secure multiinstitutional blessing, and formalize a proposal to NIH; and finally, to avoid loss of momentum the LINC team would immediately be taken in under the CDO wing to complete the redesign effort and mount a program for the dissemination of LINC technology.

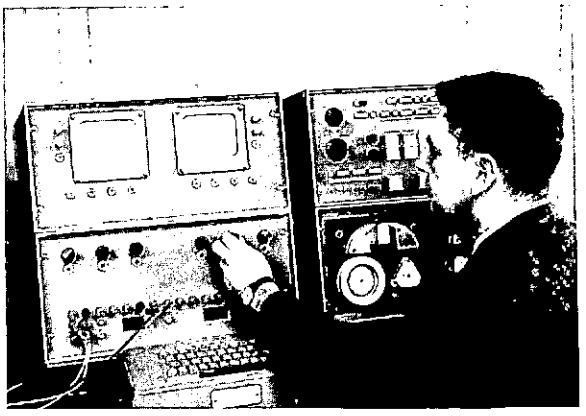
This proposal was widely accepted almost at once. Funds for the establishment of the CDO, with Walter as Director and Bill Papian as Associate Director, magically appeared. Arrangements were made with Lincoln for the transfer of staff and equipment, and the entire biomedical research computer effort at Lincoln Laboratory, together with several members of CBL, moved to new quarters next to the campus in Cambridge in January, 1963.

Redesign of the LINC

The next several months were ones of intense activity as we worked hard to complete the redesign in time for what was now formally known as the LINC Evaluation Program. Construction of 16 LINCs was authorized under the formal program. Twelve of these machines were to be assigned to biomedical research scientists selected by a nationally constituted LINC Evaluation Board for trial use in their own laboratories following a special Phase I training program at M.I.T. Four machines were to remain with the design team for coordination of Phase II evaluation activities and further refinement of the instrument itself. Given the new CDO framework Dr. Waxman of the NIH Committee, in a dazzling display of civil service at its best, had been able to earmark about half of the needed funds from NIH and had persuaded Dr. Orr Reynolds of NASA to put up the remainder from NASA's recently established Bioscience Program. In parallel with this activity NIH assembled the necessary committees to review the much larger proposal for the Center itself, and with amazing smoothness and speed authorized a huge M.I.T. research grant of an unprecedented scale sufficient to warrant announcement on the front page of the New York Times.

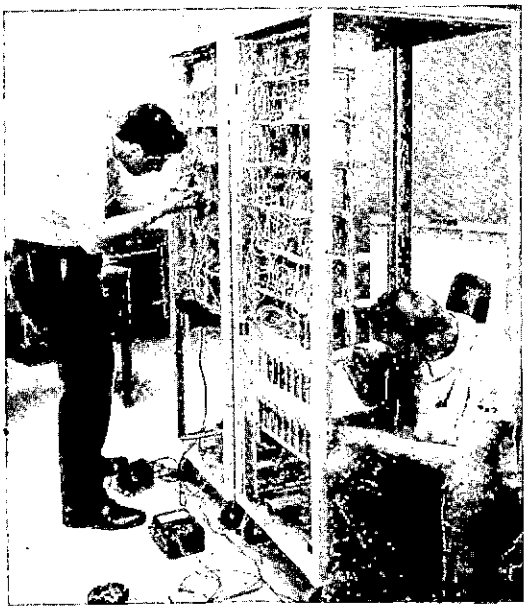
The idea of providing replicas of the LINC to individual researchers seemed quite natural. Tom Sandel had outlined just such a plan following the summer workshop the previous year and the idea had taken hold immediately. Charlie had then suggested, only partly in jest, that the computers be

put together by the workshop participants themselves and I had promptly taken the idea quite seriously as an excellent way to teach the inner workings of the LINC. After all, it would be up to the participants to keep their machines running, wouldn't it? What better way to ease these good people into the discipline of digital systems? We would gather the parts and subassemblies together in the form of a kit and provide assembly instructions and any necessary documentation. This would not only keep the costs down but would also encourage later participation by other interested individuals.

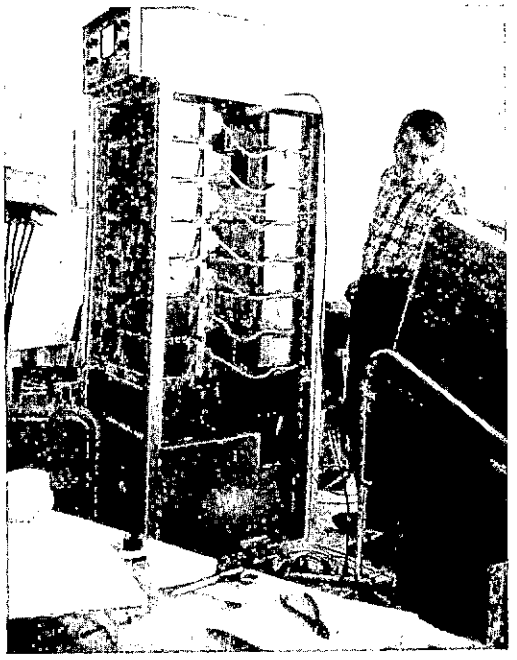


William Simon with an interim version of the LINC. The dual-display module provided both long- and short-persistence phosphor CRTs. Six potentiometer knobs supplied input parameters.

Again the projected time schedule was extremely tight, especially for a development group in an academic setting. But morale was very high as it always is under such circumstances and every member of the now somewhat expanded design team proceeded to put in a magnificent performance. Charlie designed new circuits for the memory (now modularized in two 1K units with only one to be included initially) and completely reworked the magnetic tape unit read/write amplifiers and motor control circuits. Bill Simon redesigned the a-to-d circuitry and worked out a new single-scope system for the display module. Severo expertly refined the tape control logic and incorporated the tape pre-marking subsystem. Mary Allen traveled back and forth to the TX-2 at Lincoln Laboratory to update the LINC simulator and complete the new LINC Assembly Program, LAP3. Don Malpass joined us temporarily from Lincoln and did an outstanding job of designing the power supply and the interconnection system, the high-speed computer's Achilles' heel if not very skillfully handled. Norm Kinch prepared and endlessly updated the necessary drawings, and supervised our support staff as well as the beginning manufacture of subassemblies by local vendors. I concentrated on the surprisingly difficult job of working out the details of console control and the proper arrangement of indicators and switches, and generally tried to keep the overall program on target.

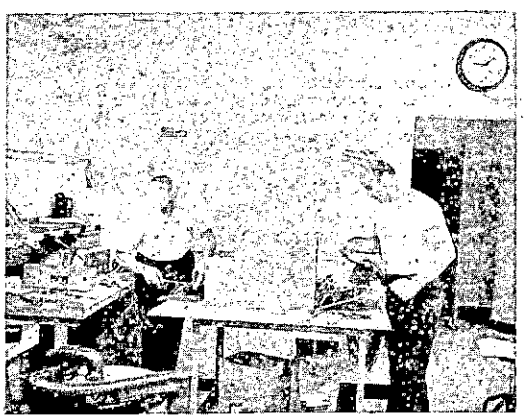


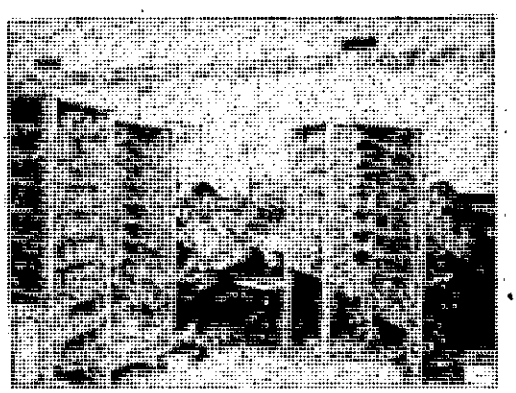
Technicians assembling wired frames for the LINC electronics cabinet. About twenty of these cabinets were required for the summer program.



Charlie debugging a prototype of the final LINC system. Modifications to the frame wiring were few but had to be made to all frames being assembled by a local vendor.

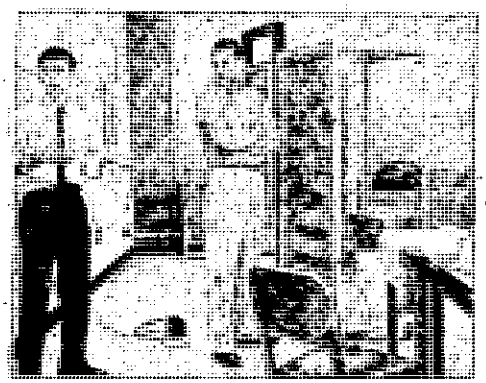
One month to go. Parts and subassemblies were continuing to arrive and the final LINC prototype neared completion. Mary Allen finished off the layout of the central control logic. Mishell Stucki, a man without peer in attention to detail, organized the wiring lists in a form suitable for the electronics frame checkout so critical to the success of the kit concept. Mort Ruderman of DEC took personal responsibility for assuring that sets of DEC modules, about 90% of the required electronics, would arrive on time. Howard Lewis and Dan Calileo and others stepped up the pace of special parts assembly under Norm and Charlie's supervision. Don O'Brien saw to it that the extensive and growing documentation was kept in good shape and had the assistance of Henry Littleboy of Massachusetts General Hospital. Severo was everywhere. One could not have hoped for a more dedicated and competent team.

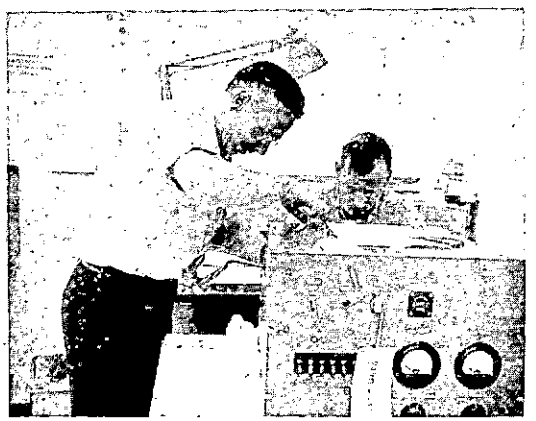




(Upper) Norm Kinch (left) and Dan Calileo repairing prototype power supply; (lower) Howard Lewis and Dan making "final"-wiring changes to the electronics frames. Several additional modifications were later found to be necessary and were added, using white wires to avoid confusion with color-coded frame wiring already documented.

[Phase I photographs taken by Bill Simon.]







(Top to bottom) Severo Ornstein (left) and Prof. Jerome R. Cox of Washington University review last-minute problem; Bob Brown (left) and Tom Sandel examine incoming power-supply; I check the wiring of a control panel module.

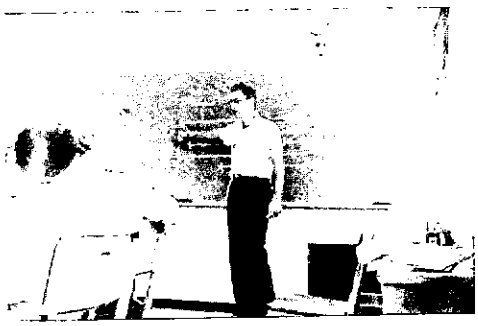
One week to go. One or two frames had been completely wired but there was still difficulty in obtaining the necessary test equipment to verify wiring accuracy. A last minute circuit problem that Charlie had discovered was overwhelmed with the help of Maynard Engebretson and Prof. Jerome R. Cox (a member of the Evaluation Board), who made

a special trip from Washington University in St. Louis. Tom Sandel, Chairman of the LINC Evaluation Board, monitored progress with increasing anxiety. Charlie and I maintained a facade of manifest confidence that masked serious concern.

Phase I: The LINC Summer of '63

On July 1 the first group of visitors arrived, some of them with golf clubs they would never use. We immediately put them into a crash course on the theory and use of computers, a holding action managed by Mary Allen and Irving Thomae and Severo for the two more weeks it would be until the kits were ready.







(Top to bottom) Severo explains LINC logic to the first group of visiting scientists; Irving Thomae reviews timing diagrams; Mary Allen Wilkes shows LINC programming.

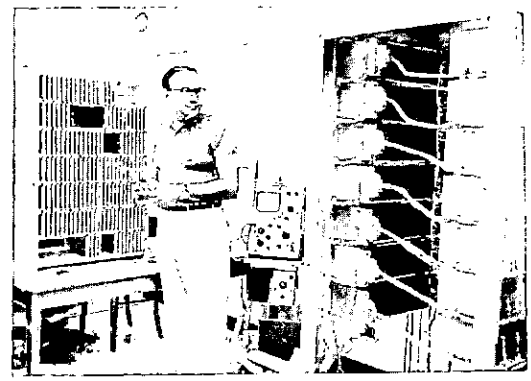




(Upper) Tom Sandel (right) in discussion with Dr. Gerhardt Werner of Johns Hopkins University; (lower) Mary Allen works on a program using rack-mounted equipment.

While classes were proceeding, the rest of us worked over the electronics frames and completed the job of checking the wiring, using Mishell's well-organized lists. The only part of this miserable and boring task we found rewarding was a sparkling Christmas-tree light effect produced by the test equipment. We found only a few wiring errors and fixed them.

Finally the assembly process began. To our delight and relief all went smoothly, though many long days and nights of exhausting work were still required to make up the lost time and correct minor design mistakes as we found them. The visiting scientists energetically and with much spirit and enormous dedication to the task proceeded quite successfully to bring all six or so LINCs to life.





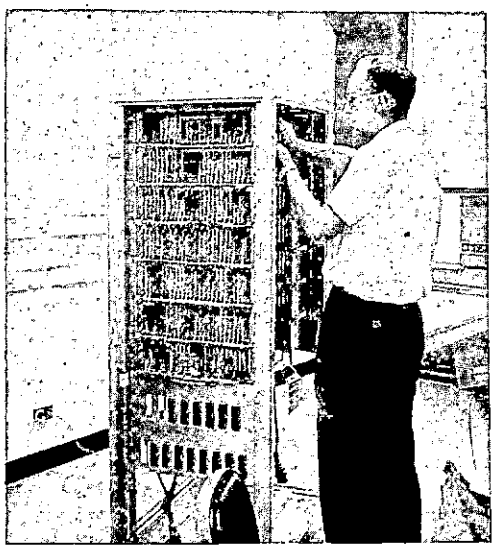






(Upper) Dr. A. J. Hance examines his tape unit module. Tape unit modules were almost entirely electromechanical; one critical component was a rubber-band. (Lower) A group of visiting scientists mount display units in staging racks.

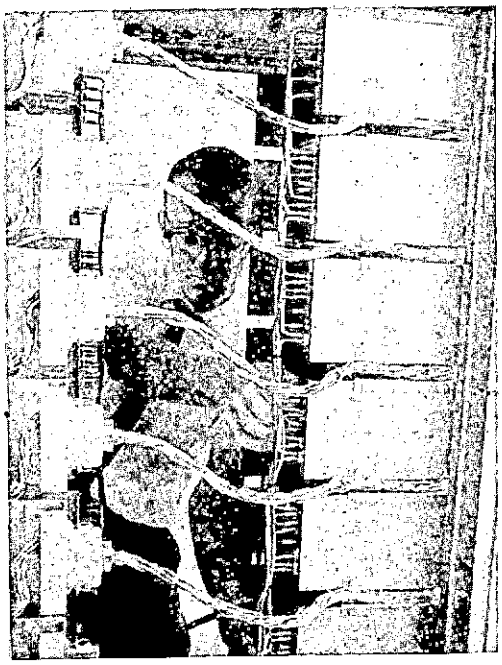
(Top to Bottom) Dr. G. F. Poggio contemplates the plug-in unit installation task; Drs. N. Bell (left) and R. Stacy with staging rack; Dr. J. C. Lilly installs a modification while Mishell Stucki (rear right) ponders a problem.

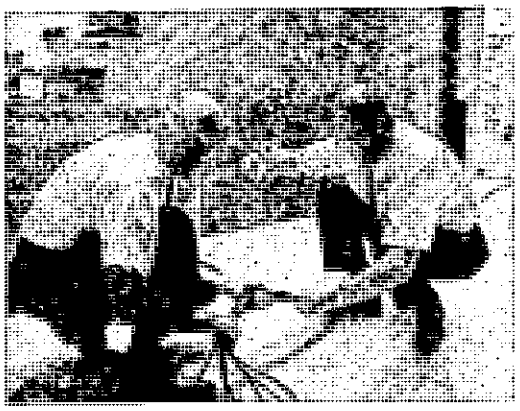






(Top to Bottom) Dr. J. W. Woodbury installs a plug-in unit; Charlie discusses a point with Dr. K. Killam (right) while Dr. Hance nods (it is 2 a.m.); Drs. C. D. Geisler (left) and J. E. Hind examine wiring.





(Upper) Dr. C. A. Boneau reaches into his cabinet to make an adjustment. (Lower) Drs. F. S. Grodins (left) and J. E. Randall wrap interconnection cables.

Each of the visiting teams was required to write and demonstrate a small program representative of the work of its home laboratory. With the completion of these final assignments, the machines were packed up for shipment and the entire group of hosts and visitors, now fast friends, celebrated with great exuberance at a farewell banquet.

The M.I.T. hosts had only three days to recover before the second group of visitors arrived. This final session of Phase I went more smoothly than the first, with less pressured teaching and more time for thoughtful assembly. Once

again the visitors carried out all of their assignments with flair and great good humor, and again — on time — the LINCs were packed up for shipment. Our final banquet this time was an occasion for sober reflection on the magnitude and importance of what we had all achieved.



Moving day. Although some parts were crated for transportation, the electronics cabinets were simply taped up and carried in moving vans as furniture. Most of the machines were moved in this manner and all arrived safely at their new homes in laboratories across the country.

The LINC was now well and safely launched and in good hands. Charlie and I had little doubt of its ultimate success.

For the most part the participants had all taken to the unfamiliar fairly well. For those few who had already worked with computers, the general acceptance of machine-level programming as the price of dealing with such a small memory had been cheerful enough. Productive software, however, would not begin to accumulate in the various laboratories until the following year; the first published version of the programming manual, the year after that; and the first edition of the final assembly program, LAP6, not until 1967.

We had had to formulate both teaching and assembly procedures as we went along, but this resulted in only a few bruised knuckles and egos and a few inefficient hours (one of the visitors, a neurosurgeon, did have difficulty adjusting a tiny slotted-screwhead potentiometer deep within the recesses of the electronics frame, as I recall). Fixing design mistakes and modifying the circuits and wiring to reduce system noise meant repairing all machines in parallel. Each machine, by the way, had required a budget outlay of about $32,000, or so the accountants told us. Not too bad a miss.

Throughout it all we had been sustained in our endeavors not only by our boundless energy and enthusiasm, our spirited sense of adventure, and our deep reserves of unshakeable conviction and commitment, but also by the Fox & Tishman Restaurant of Kendall Square, Cambridge. The F&T's sympathetic management and colorful staff fed and otherwise refreshed us at all hours of the day and night. [Following a suggestion of Charlie's, any member of the team who was responsible for a design goof agreed to buy one martini for all other members of the team at the end of the effort — one martini per goof, non-cancelling. On the day of reckoning we all repaired, of course, to the F&T. I can't remember how it all turned out.]

The Move to Washington University

The weeks immediately following our intensive summer effort did not go well. We were not only exhausted but were also confronted with an unanticipated organizational problem. The nascent M.I.T. Center, prospectively multi-institutional and multi-disciplinary, turned out to be irremediably multi-problematical as well and de-materialized. The result was that once again the peripatetic LINC team found itself in need of a more suitable home for its work. We were ...um... disappointed.

I suppose that these days it is natural to expect a group in such circumstances to form a commercial company. The LINC team, however, had always been more academically than entrepreneurially inclined. In many subsequent months of searching, the principals traveled about the country and met with many university presidents and trustees, examining several propositions quite seriously. But it was an unscheduled meeting in Cambridge with George Pake that convinced us that the best choice for the continuation of the program was Washington University in St. Louis, where Dr. Pake was Provost in the years before he left the academic world to set up XEROX PARC. He had heard about our situation from Professor Cox, he said, and had "just happened" to be visiting Cambridge on his way to Woods Hole (he never got there). He had stopped by so that we could, in his words, look him over. We were already well acquainted with Jerry Cox, an old friend even then, especially since those final hectic days before summer, and had a high regard for his innovative work in biomedical computing at Washington University. For many of us, then, this extraordinary meeting with George Pake made an already very promising possibility irresistible.

And so it was that the Washington University Computer Research Laboratory was established in 1964 under the direction of William N. Papian, Associate Dean of Engineering. CRL set up shop in newly renovated space made available by its newfound "sister laboratory", the Biomedical Computer Laboratory under Jerry Cox. Tom Sandel accepted an appointment as Professor of Psychology, and I became Research Professor of Computer Science and Associate Director of CRL. Many from the Cambridge team, including Severo, Mishell, Howard and Ken Lewis, and Constance and Joe Towler uprooted and moved to St. Louis, followed by others from M.I.T. over the next few years. Charlie remained in Cambridge to complete both his military service and his doctorate at M.I.T. and rejoined the group a

year later as Associate Professor of Physiology and Biophysics. Norm Kinch rejoined us after an assignment at an experimental radar site operated by Lincoln Laboratory in the Southwest. Mary Allen received her appointment letter from George Pake midway in a one-year trip around the world (she was then in Calcutta), but before coming to St. Louis spent another year as an extramural member of CRL working on a new assembly-program/operating system, LAP4, using a LINC set up in the living room of her family home in Baltimore — surely a first in personal computing. But the original team never entirely regrouped. To our great regret, George Gerstein and Belmont decided that their immediate scientific interests were better served by accepting appointments at the University of Pennsylvania in the Laboratory of Dr. Britton Chance, and Bill Simon elected to accept an appointment at the Harvard Medical School.

Phase II Activities

Four main tasks faced us in carrying out our Phase II responsibilities under the formal evaluation program.

1) We had to assure that the LINC equipment itself had been transplanted successfully into each of the participants' laboratories and was kept up-to-date.

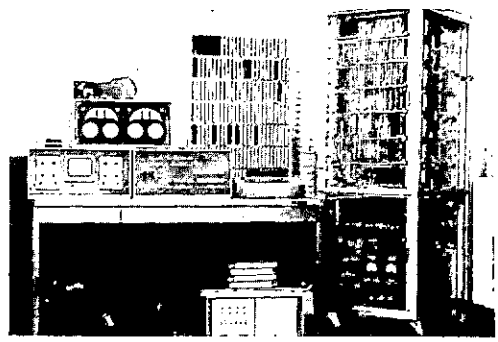
2) We had to complete and distribute further documentation on the machine and its use.

3) We had to improve LINC software, starting with an enhanced version of the basic assembly program that provided automatic filing capabilities and improved editing and operational features.

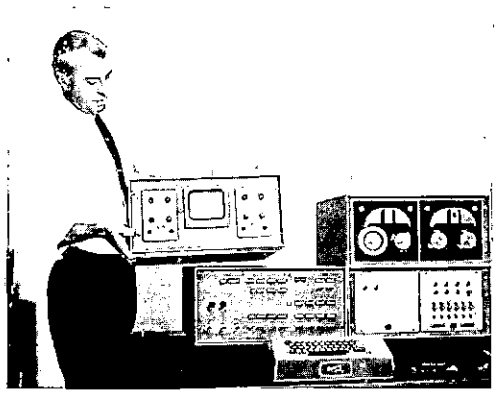
4) Finally, we had to accumulate operating experience with the LINC in our own work in biomedical research and in other application areas.

Assuring ourselves and our sponsors that the LINC was taking root required considerable use of the telephone and post office as well as visits to each site. A few faulty switches and parts needed to be replaced, a few adjustments made, and a few circuit improvements installed as we verified their need. We adhered to the principle that any modification to one machine meant a modification to all machines. By and large, however, machine reliability was excellent, thanks to sound engineering by Charlie, Bill Simon, and Don Malpass as well as to the robustness of DEC circuit packages when carefully used. [From one of the sites, the only call for hardware help we ever got dealt with what to do about the fact that the elapsed-time meter on the power supply had just jammed at 99999.]

Completing a set of finished documents on LINC hardware and operation was more difficult (the evaluation program participants had been able to take away from Cambridge only the roughest of descriptive material and operating notes). Among other things, it meant that a full production version of a LINC kit and its accompanying assembly and test procedure documentation had to be checked out and certified. Not long after CRL had been set up, Mort Ruderman brought the first complete DEC-produced kit to St. Louis. Mishell and Severo then put it together and verified the accuracy of the documents and test procedures, making a few necessary corrections. Final documents were prepared with the help of newcomers Dave Stewart (who had worked with Charlie during his Air Force days), Gerald Johns, Robert Ellis, and Maurice Pepper. After printing and distributing revised final documents to all evaluation program participants, we sold document sets at cost to anyone who wanted them.







(Top to bottom) The LINC kit produced by DEC; Mort Ruderman of DEC (left), Mishell (rear) and Severo assembling the kit; Norm Kinch stacks console modules.

Building up a base of operating software was accelerated with the help of the Biomedical Computer Laboratory. A variety of utility programs were written by our BCL colleagues Jerry Cox, Mike MacDonald, Sharon Davisson, and others. BCL incorporated these into LAP4 (an interim modification of the LINC Assembly Program, LAP3), and many of its operating features were later incorporated by Mary Allen in the ultimate version, LAP6. Software was generally distributed on LINC tape reels via the mails. Since direct exchange of programs among the participants was rare, we did what we could to certify and distribute copies of application software that was submitted to CRL or written at CRL or its sister laboratory BCL.

Use of the LINC in our own work was supplemented by work at BCL, where a few kits were assembled and used in connection with collaborative programs with the Washington University School of Medicine and its associated hospitals. Later we acquired several of the LINC variant machines made by DEC and others and began to use them in our developing research program in macromodular systems (which soon took over as the major focus of CRL activities). [BCL went on to design the Programmed Console, the result of a design seminar that Jerry and I taught in 1965 [17]. Known as the PC, it provided both autonomous and remote-connnection modes of operation and thus was one of the earliest "smart terminals". It was subsequently produced in small numbers and made available to radiologists for radiation treatment planning under a national program modeled after the LINC Evaluation Program.]

The Final Evaluation Program Meeting

The program participants began to use and then gradually depend on the LINC in their research. It had not seemed so, however, at our first regathering in June, 1964, when we all met for a show-and-tell at a distinguished old resort hotel in New Hampshire. Just short of a year since the Cambridge Summer, there hadn't yet been enough experience gained in the use of the new tool, especially in view of the scarcity of software and the upsetting problem of relocating the LINC core group. No one was greatly surprised. But at the final LINC Evaluation Program meeting in St. Louis in March, 1965, the participants were well prepared and had by then accomplished a great deal of scientific work. The studies reported covered a wide range of topics [18]. Some of them were:

Operant conditioning of pigeons and monkeys. [One series of experiments made use of a wonderful Lazy Susan holding a pigeon at each of four stations around its circumference, with indexing into data-collecting position, stimulus generation, and data analysis all provided by the LINC.]

Spontaneous activity in thalamic and cortical neurons, behavior of single cochlear neurons, and somatosensory responses evoked in intracranial stimulation of cortex.

Pulsatile blood flow, cardiac muscle behavior, hydrodynamics of the mammalian arterial system, and human finger tremor.

Genetics at the bacterial and molecular level using mass spectrometers and other instruments.

Communication between man and dolphin. [An unanticipated equipment hazard: the dolphin, one Elvar, could accurately spit a stream of seawater a distance of twenty feet.]



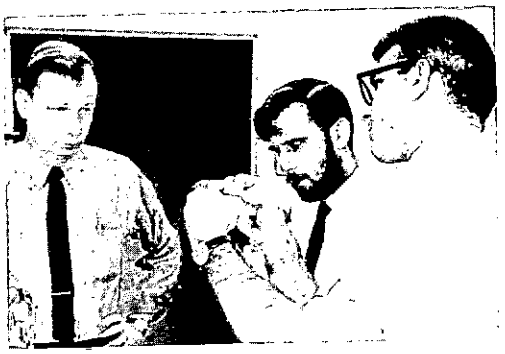


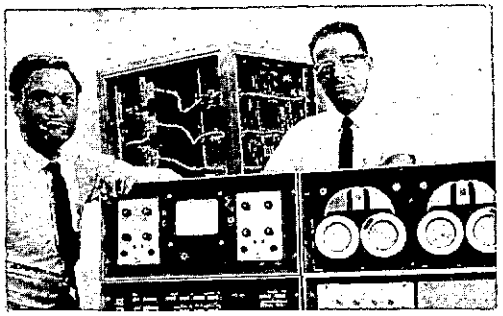


Preparation for the Final LINC Evaluation Program Meeting, March, 1965. (Top to bottom) I talk with Joseph Foley of Spear, Inc. while a LINC is being set up on stage; Prof. Harold Shipton (left) in discussion with Mort Ruderman and Winn Hindle of DEC; Drs. L. Wienckowski (left) and Bruce Waxman of NIH.

In addition to their scientific accomplishments the participants reported several other technical and software development activities. These included the connection of the LINC to computation center machines, the development of a system for remote control of the LINC, the design of interface circuits for control of a blue-format tape transport and other laboratory equipment, and the development of several experimental operating system programs and floating-point routines.

By the end of Phase II the Evaluation Program participants had an accumulated experience of about 50,000 hours of LINC use with remarkably little trouble, and at least that many hours had also been accumulated at other LINC sites with similar results. For the most part the original participants were awarded permanent custody of their machines following their reports of this strikingly successful operation in laboratories across the country.





(Upper) Tom Sandel and a now bearded Severo in a visit to the laboratory of Dr. G. S. Malindzak; visits were made by various members of the Washington University group to each of the participating Evaluation Program laboratories. (Lower) The author and William N. Papian with the final version of the LINC [Photo courtesy of the Saint Louis Post-Dispatch].

IN LATER YEARS

Although our formal responsibilities under the LINC Evaluation Program had been discharged, there was still work to be done that would take some of us a few more years.

Final LINC Modifications

Following the recommendation of the growing band of LINC users as reported at the final Evaluation Program meeting in St. Louis, a decision was made to undertake a number of hardware modifications. A new register and an arithmetic overflow circuit were provided to extend the range of arithmetic and to facilitate running a certified floating-point software package for which a considerable consensus had developed among the mathematically inclined. These changes were seasoned, documented, and distributed (with a few installation visits) to all of the original LINC sites, as was the floating-point package itself. Similarly, data and program interrupt features were incorporated to improve performance in several high rate applications. Altogether six new instructions dealing with these new features were wired into all machines.

The Final LINC Operating System

The final assembler/operating system, LAP6, was put together by Mary Allen with great care. Discussion of its specification had begun early in 1965 during her visits to several of the participating laboratories to develop a consensus regarding users' needs. A decision to double the initial 1K memory size had already been made; but as Mary Allen put it, doubling a 1K memory produces another small memory. Despite this limitation she was able to define a remarkably efficient operating system based on a LINC tape scrolling algorithm developed by Mishell and Severo. LAP6 included a line-by-line editor, an assembler, automatic management of files, user-defined metacommands, and runtime debugging aids [19]. It was thoroughly tested before being distributed in 1967 and the accompanying LAP6 Handbook still stands as a model of simplicity, accuracy, and usability [20]. A student of philosophy (today a lawyer in Massachusetts), Mary Allen quoted Kierkegaard in the foreword to the handbook:

To promise the System is a serious thing.

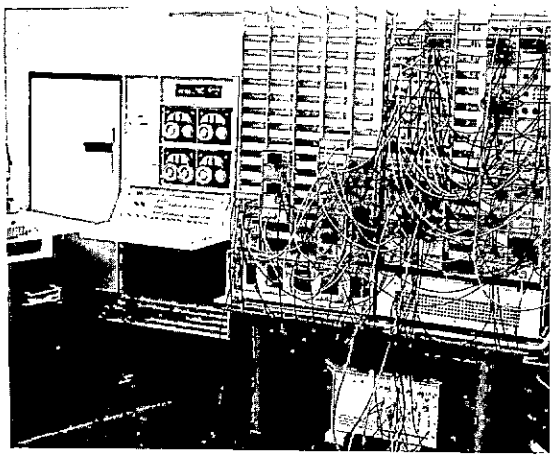
The handbook and the second edition of its companion programming tutorial document published in 1969 reflect the final form of what is now referred to as the "classic" LINC [21].

Manufacture of the LINC and LINC Variants

DEC went on to manufacture altogether fifty or sixty classic LINCs, only a few of these in kit form — an idea whose time never quite came. [Some of the kits were assembled for others by Prof. Harold Shipton of the State University of Iowa, previously a member of the LINC Evaluation Board. "Shippy", one of the most charming people ever to walk the face of the planet and well known to the biomedical community for his perceptive and imaginative engineering, later joined the faculty of Washington University.]

In 1965 I joined forces with Dick Clayton of DEC to design the LINC-8, a 32K dual-machine combination of the LINC and Gordon Bell's PDP-8, in which the complex LINC tape control logic was relegated to a concurrently executable program on the PDP-8 side using software written by Severo. While the nearly 150 LINC-8s were being made, Dick designed a new version of the machine, the PDP-12, of which about 1000 were manufactured [3].

In 1965 a new Massachusetts start-up company, Spear Inc., made its appearance. It began to manufacture a 4K integrated circuit version of the LINC, called the micro-LINC 1. Spear went on to produce the micro-LINC 300, a higher-performance 32K variant. [Spear, Inc. subsequently became a division of Becton Dickinson and Company. I have no doubt that the transaction helped Spear's former president farther along toward his life-long goal of retiring to some small South Sea Island that had an 18-hole golf course and a native monarch.]



A Spear Inc. micro-LINC 300 (left) providing operational support to an experimental macromodular system at Washington University.

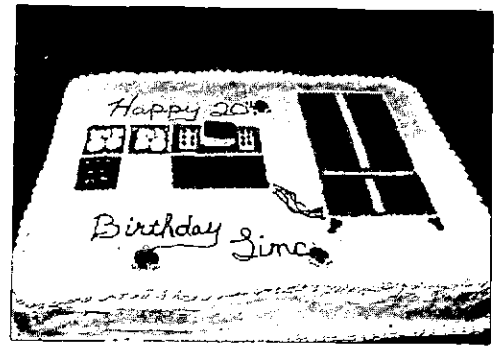
DENOUEMENT

With more than 1200 LINC or LINC variants at work around the world, research over the ensuing years began to result in very substantial contributions to the body of scientific literature. An effort was made to maintain a bibliography of LINC-related publications for the first few years, but the task soon got out of hand. The July, 1969 edition lists more than 150 scientific papers, journal articles, and published books [22]. By then the growth rate of bibliographic references was already doubling each year, and the number of LINC programs and application notes alone soon became so large that responsibility for handling them had to be turned over to the DECUS Program Library.

The careers and lives of those associated with the LINC development continued to grow and change in many ways, as new academic appointments were assumed and new professional challenges undertaken. Many early students of these scientists and engineers, and their students in turn, went forth to seek their fortunes after training in the LINC and its ways.

Washington University's program in the medical and biological research uses of computer science and technology grew in prominence and soon came to be regarded among the foremost in the country. In the course of years Prof. Jerome R. Cox became Chairman of the Department of Computer Science, and Prof. Charles E. Molnar, Director of the Institute of Biomedical Computing. Dean William N. Papian retired from the University to private consulting practice, as did I. In a very saddening turn of the historical wheel Prof. Thomas T. Sandel, after many years as Chairman of the Washington University Department of Psychology, died just weeks before the LINC's twentieth anniversary gathering of old friends of the Summer of '63 was to take place.

The LINC design went on to influence the design of DEC machines that further reinforced the concept of the small computer as a tool for personal work and very soon took over center stage. Yet over the years the LINC and its variants continued to serve the needs of biomedical research. A few of them were still on the job productively in 1985, and one of these — perhaps the last of the "classics" — could be found at the Massachusetts General Hospital, not yet retired after more than two decades of continuous use.



The birthday cake that made its appearance at the LINC Twentieth Anniversary Celebration held on November 30, 1983 at NIH.

References

[1] Clark, W. A. and C. E. Molnar: A Description of the LINC. in *Computers in Biomedical Research I*. R. W. Stacy and B. D. Waxman (eds) Academic Press, New York, 1965; ch 2.

[2] Redmond, K. C. and T. M. Smith: *Whirlwind - The History of a Pioneer Computer*. Digital Press, Bedford, Mass., 1980.

[3] Bell, C. G., J. C. Mudge, and J. E. McNamara: *Computer Engineering - A DEC View of Hardware Systems Design*. Digital Press, Bedford, Mass., 1978.

[4] Farley, B. G. and W. A. Clark: Activity in networks of Neuron-like Elements. in *Information Theory, Fourth London Symposium*. C. Cherry (ed), Buttersworth, London, 1961: 242-251.

[5] Clark, W. A.: Design Considerations for an Experimental Computer. Linc. Lab. memo 6M-3536, April 1955.

[6] Clark, W. A.: The Multi-Sequence Concept. Linc. Lab. memo 6M-3144, November 1954.

[7] Forgie, J. W.: The Lincoln TX-2 Input-Output System. *Proc. 1957 WJCC*: 156-160.

[8] Thacker, C. P. *et al.*: ALTO: A Personal Computer. in *Computer Systems: Principles and Examples*. Siewiorek, Bell, and Newell, (eds), McGraw Hill, 1982; ch 33.

[9] Mitchell, J. L. and K. H. Olsen: TX-0, A Transistor Computer with a 256 by 256 Memory. *Proc. 1956 EJCC*. 10: 93-101.

[10] Clark, W. A.: The Lincoln TX-2 Computer Development. *Proc. 1957 WJCC*: 143-145.

[11] Clark, W. A., R. M. Brown, M. H. Goldstein, C. E. Molnar, D. F. O'Brien, and H. E. Zieman: The Average Response Computer (ARC): A Digital Device for Computing Averages and Amplitude and Time Histograms of Electrophysiological Responses. *IRE Trans. Biomed. Electronics*, 1961, BME-8:46-51.

[12] Clark, W. A.: A Functional Description of the L-1 Computer. Linc. Lab. Rept. 51G0012, ASTIA 236678, 1960.

[13] Clark, W. A.: Digital Techniques in Neuroelectric Data Processing. in *Computer Techniques in EEG Analysis*, M. A. B. Brazier, ed. EEG J. supp. 20, Elsevier, 1961.

[14] Best, R. L. and T. C. Stockebrand: A Computer-Integrated Rapid-Access Magnetic-Tape System with Fixed Address. *Proc. 1958 WJCC*.

[15] Clark, W. A.: The General Purpose Computer in the Life Sciences Laboratory. in *Engineering and the Life Sciences*. NAS-NRC Rept., Wash. D.C., April 1962; 33ff.

[16] Rosenfeld, S.: LINC - The Genesis of a Technological Revolution. *LINC 20th Anniversary Seminar*. NIH Rept. November 30, 1983.

[17] Cox, J. R.: A Description of the Programmed Console. BCL Monograph #37, Wash. Univ., September 1966

[18] *Convocation on the Mississippi: Final Report of the LINC Evaluation Program*. Wash. Univ., 1965.

[19] Wilkes, M. A.: Conversational Access to a 2048 Word Machine. *Comm ACM*. 13:July 1970; 407-414.

[20] Wilkes, M. A.: *LAP6 Handbook*. CRL Tech. Rept. #2, Wash. Univ., May 1967.

[21] Wilkes, M. A. and W. A. Clark: *Programming the LINC*. Wash. Univ., January 1969.

[22] Wilkes, M. A.: Bibliography of LINC-Related Publications. CSL Ref. Lib. LINC Doc. #14, Wash. Univ., January 1969.

APPENDIX

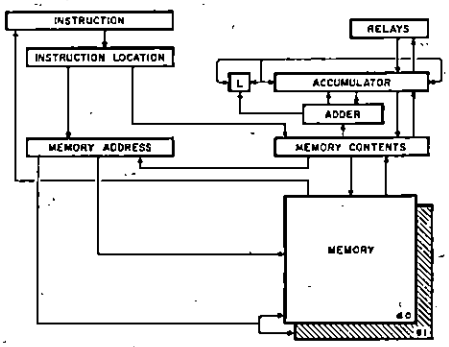
The LINC Evaluation Program Participants

Without any doubt whatsoever, as Tom Sandel would have said, the success of the LINC is due in incalculably large part to the pioneering work of the evaluation program participants. Its formal members were: Drs. E. O. Attinger and A. Anne, Research Institute, Philadelphia Presbyterian Hospital; Dr. D. S. Blough, Dept. of Psychology, Brown University; Drs. S. Goldring and J. L. O'Leary, Dept. of Neurology, Washington University; Drs. F. S. Grodins and J. E. Randall, Dept. of Physiology, Northwestern University; Drs. J. E. Hind and C. D. Geisler, Laboratory of Neurophysiology, University of Wisconsin; Dr. J. Lederberg and Lee Hundley, Dept. of Genetics, Stanford University; Dr. C. A. Boneau, Dept. of Psychology, Duke University; Dr. J. C. Lilly, Communication Research Institute; Drs. G. S. Malindzak and F. S. Thurstone, Dept. of Physiology, Bowman-Gray School of Medicine; Drs. G. F. Poggio, G. Werner, and V. B. Mountcastle, Dept. of Physiology, and Dr. B. Weiss, Dept. of Pharmacology and Experimental Therapeutics, Johns Hopkins University; and Drs. J. W. Woodbury and A. M. Gordon, Dept. of Physiology, University of Washington.

The number of LINCs actually built in the summer of 1963 was about twenty. Several other investigators had been invited to participate on an informal basis. These informal participants assembled their computers alongside the others, using their own funds but taking advantage of whatever batch orders were placed for parts and subassemblies. They were: Drs. K. Killam and A. J. Hance, Dept. of Pharmacology, Stanford University; Dr. H. E. Tompkins and Mr. J. S. Bryan, National Institutes of Health; Dr. J. B. Lewis, M.I.T. Lincoln Laboratory; and Drs. R. Stacy and N. Bell, Institute of Statistics, North Carolina State College.

Under the chairmanship of Dr. T. T. Sandel of

M.I.T., the LINC Evaluation Board included Dr. M. A. B. Brazier of the University of California at Los Angeles, Dr. J. R. Cox of Washington University, Dr. E. R. Dempster of the University of California, Dr. J. Macy, Jr., of the Albert Einstein School of Medicine, Dr. H. V. Pipberger of the Veterans Administration, Dr. M. D. Rosenberg of the Rockefeller Institute, and Prof. H. W. Shipton of the State University of Iowa.



THE EVALUATION PROGRAM LINC INSTRUCTION SET

FULL ADDRESS CLASS — ADD | X

The right-most ten bits of the instruction word specify one of the memory registers 0-1777 (octal).

ADD X	Add the contents of memory register X to the Accumulator.
STC X	Store the contents of the Accumulator in memory register X, and clear the Accumulator.
JMP X	Take the next instruction from memory register X, and save the subroutine return point in memory register 0.

INDEX CLASS — LDA | i | β ; LDA | i | 0 → OPERAND OR ADDRESS

If $\beta \neq 0$, the instruction is single word length and the address of the memory operand is located in index register β ($1 \leq \beta \leq 17$ octal). If $\beta = 0$, the instruction is double word length and the second word is either the operand (case $i = 1$) or the address of operand (case $i = 0$). The specified index register is incremented by ONE before use if $i = 1$.

LDA i β	Load the Accumulator with the memory operand.
STA i β	Store the contents of the Accumulator in the specified memory location.
ADA i β	Add the memory operand to the contents of the Accumulator.
ADM i β	Add the memory operand to the contents of the Accumulator and leave the sum in the Accumulator and in the memory operand location.
LAM i β	Add the memory operand to the contents of the Accumulator, together with any previous overflow held in the Link Bit. Leave the sum in the Accumulator and the memory operand location, and retain any new overflow in the Link Bit.
MUL i β	Multiply the contents of the Accumulator by the memory operand, and retain either high-order half or low-order half of the double-length product in the Accumulator.
SAE i β	Compare the contents of the Accumulator with the memory operand, and skip the next instruction if they are identical.
SRO i β	Rotate the memory operand to the right one place and skip the next instruction if the sign bit of the operand is ZERO.
BCL i β	Clear each bit in the Accumulator which corresponds to a ONE in the memory operand.
BSE i β	Set to ONE each bit in the Accumulator which corresponds to a ONE in the memory operand.
BCO i β	Complement each bit in the Accumulator which corresponds to a ONE in the memory operand.
DSC i β	Display character intensifying points on the scope in a 2 by 6 array according to the bit pattern of the memory operand.

HALF-WORD CLASS — LDH | i | β ; LDH | i | 0 → OPERAND OR ADDRESS

Addressing is similar to that of the INDEX CLASS, except that the left-most bit of the address specifies which half of the memory operand is used. When $i = 1$ and $1 \leq \beta \leq 17$ (octal), the address word in index register β is incremented before use, in such a way as to step through consecutive half-words, the address of the operand increasing by one on every second reference.

LDH i β	Load the Accumulator with the specified half of the memory operand.
STH i β	Replace the specified half of the memory operand with the contents of the right half of the Accumulator.
SHD i β	If the contents of the right half of the Accumulator are not identical to the specified half of the memory operand, skip the following instruction.

TAPE CLASS — (MTP) | i | u | RDC ; SECTOR | BLOCK NUMBER

These are double word length instructions in which the first word specifies the tape unit (u-bit), the motion state following execution (if $i = 0$, the selected unit is stopped), and in which the second word specifies one of eight sectors of memory and one of 512 tape blocks between which transfers are to be made. All required searching operations and checking are performed automatically, and are repeated if checks indicate improper transfer.

RDC i u Q, N	Read tape block N into memory sector Q, and check the transfer.
RCG i u Q, N	Read from one to eight consecutive tape blocks, beginning with block N, into memory and check all transfers.
RDE i u Q, N	Read tape block N into memory sector Q.
MTB i u Q, N	Move toward tape block N on specified unit.
WRC i u Q, N	Write contents of memory sector Q in tape block N and check the transfer.
WCG i u Q, N	Write from one to eight consecutive memory sectors on tape, beginning with block N, and check all transfers.
WRI i u Q, N	Write contents of memory sector Q in tape block N.
CHK i u O, N	Find tape block N and leave the check sum for this block N in the Accumulator.

SHIFT CLASS | ROL | i | n |

If $i = 1$, the Link Bit is coupled into the shift path. $0 \leq n \leq 17$ (octal)

ROL i n	Rotate the contents of the Accumulator n places to the left.
ROR i n	Rotate the contents of the Accumulator n places to the right.
SCR i n	Multiply the contents of the Accumulator by 2^{-n}.

SKIP CLASS | SNS | i | n |

Skip the next instruction if the specified conditions are met. If $i = 1$, the skip conditions are reversed.

SNS i n	Check whether Sense Switch n is up, $0 \leq n \leq 5$.
AZE i	Check whether the Accumulator contains ZERO.
APO i	Check whether the contents of the Accumulator are positive.
IBZ i	Check whether either tape unit is reading an interblock zone mark.
SXL i n	Check whether External Level input line n is negative, $0 \leq n \leq 14$ (octal).
KST i	Check whether a key has been struck on the keyboard.
LZE i	Check whether the Link Bit contains ZERO.

SET | SET | i | β |

OPERAND OR ADDRESS

SET i β	Set register β, $0 \leq \beta \leq 17$ (octal), to the value of the second word (case $i = 1$) or to the value of the operand whose address is located in the second word (case $i = 0$).

DISPLAY | DIS | i | β |

DIS i β	Intensify a point on the scope whose vertical position is given by the right-most nine bits of the Accumulator and whose horizontal position by the right-most nine bits of memory register β, $0 \leq \beta \leq 17$ (octal). If $i = 1$, increment contents of register β before use.

INDEX AND SKIP | XSK | i | β |

XSK i β	Increment by 1 the contents of memory register β (if $i = 1$) and skip the next instruction if the new contents of this register equal 1777, 3777, 5777, or 7777 (octal).

SAMPLE | SAM | i | n |

SAM i n	Convert the analog signal appearing on channel n, $0 \leq n \leq 17$ (octal), to an eight-bit binary number in the Accumulator. If $i = 0$, this instruction takes 24 microseconds; if $i = 1$, the instruction is completed in 8 microseconds although the conversion process continues for an additional 16 microseconds.

MISCELLANEOUS | HLT |

HLT	Halt the computer and sound a gong.
CLR	Clear the Accumulator and Link Bit.
ATR	Load the six Relay flip flops from the right half of the Accumulator.
RTA	Load the Accumulator from the six Relay flip flops.
NOP	Do nothing for 8 microseconds.
COM	Complement the Accumulator.

OPERATE CLASS | OPR | i | n |

These instructions form a general input-output command set whose execution is partially controlled by externally generated signals. Some functions of this instruction set are summarized below.

1. *Pausing*, with conditional restarting.
2. *Generating outputs* on any one of sixteen control output lines.
3. *Reading into the Accumulator* from either of two twelve-bit parallel sets of gates.
4. *Reading into the Memory Buffer* from either of two twelve-bit sets of gates.
5. *Clearing of A* conditional upon externally generated signals.
6. *Gating outputs* from the Accumulator to external equipment.
7. *Transferring information* from the Memory Buffer to the Accumulator under the control of external signals.
8. *Controlling high-rate inputs and outputs* between internal core memory and external equipment.
9. *Reading of keyboard.*
10. *Reading of console switches.*

Hewlett-Packard and Personal Computing Systems

Charles H. House
Hewlett-Packard

Abstract

Scientific computing systems for individuals were pioneered early at Hewlett-Packard, beginning with the 9100A Desktop Calculator in 1968. Extensions of this first machine were soon seen in Personal Peripherals, such as Printers, Tape Cartridges, and Plotters, and followed by Graphic CRT Displays. By early 1972, the Desktop unit had been augmented by a very powerful Pocket Calculator, the ground-breaking HP 35A.

This paper traces the evolution of these machines to the present day, following four main themes: 1. Scientific Desktop Computers; 2. Personal Calculators and Portable Computers; 3. Personal Peripherals; and 4. Personal Business Computers.

The underlying technologies are described, as well as the product definitions and the market development. The company organization is described in some detail, since it appears both unorthodox in the industry and particularly appropriate for certain elements of the accomplishments.

Some of the key people will be highlighted for their personal contributions, but the theme will revolve rather around the Team structure that seems responsible for the continued success in many of these product sectors.

The paper concludes with considerations about two corollary impacts of this pioneering effort: 1. The development of Data-Domain design and test methodologies; and 2. The impact of Pocket Calculators on the mathematical skills and procedures of our society.

A History of the PROMIS Technology: an Effective Human Interface

Jan R. Schultz
PROMIS Information Systems, Inc.

1. Introduction

It is perhaps unusual for a computer technology to grow out of a social movement, but that is what this story is about. A small group of medical and computer people worked together for 15 years developing a computer technology to support a medical philosophy. The grant proposal of 1967 which provided our original funding set forth the goals that were to guide us:

> We propose that: (1) the medical record, utilizing a "problem oriented" approach, be the instrument whereby the following objectives can be implemented, and that (2) the technique of record keeping described in previous publications (Weed, 1964, 1966) be the basis for the beginning realization of these goals and that (3) a real-time computer system be used to overcome the data distribution and time barriers that are insurmountable on a manual basis using conventional hospital and clinical medical records.
>
> Objectives: To develop a system that will:
>
> 1) Facilitate good patient care by making immediately available (in minutes) to the individual physician a complete, updated list of problems on any patient and by providing simultaneously, as a unit, all the data in sequence (narrative, laboratory, etc.) pertinent to any of these problems.
>
> 2) Make possible epidemiological studies and other research endeavors in terms of problems, having all the data on any given problem immediately available.
>
> 3) Make possible a medical audit whereby the standards of care being provided for a given entity (e.g. hypertension) can be rapidly assessed because of the specific orientation of all the data.
>
> 4) Make possible a business audit to assess the physical, financial and time resources that go into the solution and management of a given problem. The need for a more organized, efficient and economical approach to the management of common medical and surgical disorders may then be documented.

These remained the objectives of the project throughout its history.

I met Lawrence L. Weed, M.D., in November of 1966. Weed had published his initial paper on a new organization of the medical record: "Medical Records, Patient Care and Medical Education" (Weed 64). He was interested in exploring the implications of automating the new organization of the medical record. We decided to work together for six months and see if during that time a longer term plan of collaboration could be worked out.

When we began our work, other medical record research groups took the dictated or written words of physicians and other medical personnel and manually entered them into a computer. We decided from the very beginning to interface the information originator directly to the computer and develop techniques and tools to facilitate this direct interface. We decided on this direction for a number of reasons: among them first, the time lag between the dictation and the entry of the medical information could have a negative impact on the patient care delivered, and second, the use of the computer as a recall and structuring mechanism for medical knowledge and medical records could allow the medical personnel to

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0159 $00.75

operate from a universe that potentially could be much larger than what they could remember, and the data gathered in a specific area of the record could be consistent across all health practitioners entering it. The decision to directly interface the information originator brought us directly to the major areas of our research: the form of the human interface and the representation of medical knowledge within the computer.

Using computer technology to store and retrieve the medical record required the medical knowledge entered into the computer to have a rigorous structure. The medical community considered the notion of structuring a present illness, a medical history or a physical examination tampering with the art of medicine. Developing outlines of the structure of the various elements of the medical record was only the beginning of the effort. We had to find physicians, nurses, pharmacists, laboratory technicians, radiologists and other medical personnel who were not only organized in their thinking, but willing to expend enormous mental effort to fill in the outlines with medical knowledge to make the system usable. The medical group in the PRoblem Oriented Medical Information Systems (PROMIS) Laboratory was such a group of people.

The structure of the Problem Oriented Medical Record (POMR) served as our guide in defining the organization of all the data in the patient's medical record. The structure corresponds to four medical actions. The first is collection of a Data Base; from this the Problem List is formulated; for each active problem Initial Plans are written; and each problem is followed in Progress Notes. Without this structuring mechanism, our job would have been much more difficult because the traditional medical record has no logical structure but organizes data by its "source."

The logical POMR structure requires that the same data be retrievable in many different ways. It is necessary, within a patient's record, to look at all progress notes for a single problem or all progress notes for the last 24 hours or a tabular array of specific physiologic parameters (a flowsheet). The requirements of the POMR were beyond typical data base management systems of 20 years ago (and are beyond most of them today).

The medical record contains both narrative and numeric data. The narrative data of the medical record is variable length and the numeric data has many forms. It was impossible to define a fixed length container or set of containers that could encompass all the data in the medical record. It was necessary to manage variable length text strings that could be retrieved in different orderings to different output devices. The numeric data contained in the medical record had hundreds of forms and had to be retrieved as part of a narrative note as well as part of a flowsheet. Both narrative and numeric data needed encoded information associated with it, to facilitate the various retrievals and to make the data a legal medical record.

The initial decisions to interface the information originator directly to the computer system and to use the POMR to structure the record and provide an outline for the guidance within the computer system carried the medical and computer groups far into computer technology research and into advanced exploration of knowledge base development for the representation of medical knowledge.

2. A Search for the Correct Form for the Human-Computer Interface

2.1. Initial Tools for the Human-Computer Interface

The initial plan for the entry of medical record information was to use optical scan sheets and the IBM 1092-1093 programmed keyboards with an attached typewriter keyboard. Optical scan sheets were designed for the past medical history and systems review as well as for the physical examination. The programmed keyboards plus the attached typewriter keyboard were to be used for entry of all other information. The programmed keyboards accepted opaque plastic overlays or keymats placed over an array of 26 columns of 10 keys each, for a total of 260 labeled key locations. Each keymat could be removed and the programmed keyboard could sense the particular keymat being used, thereby identifying the type of entry into the system. Keymats were designed for each major medical subspeciality and were organized minimizing the changing of keymats and maximizing the amount of information that could be entered from the keymat without having to use the typewriter keyboard. All information entered using the programmed keyboard was printed on the attached printer so that it could be verified before being stored in the computer.

During the first six months of 1967, several thousand

past medical histories and systems reviews were processed, using two trained interviewers to fill out the optical scan sheets for entry into an IBM 1440 computer. The house staff at the Cleveland Metropolitan General Hospital were trained to fill out the physical examination scan sheets, and several hundred physical examinations were done. Great care had to be exercised in filling out the sheets or the optical scanner would reject them. The house staff had difficulty filling them out, and because of the large number of rejections, the use of the scan sheets was discontinued.

During the summer of 1967, Charles Burger, M.D., the designer of the keymats, tested the feasibility of using the programmed keyboards to enter a complete medical record. Burger's work was conclusive; the programmed keyboard presented the user with too much information, and the subsequent search time to find the proper key was excessive. The search time combined with mat changing time resulted in slow operation. A sample record took six hours to enter, at least double or triple the time required to write the original record. Scan sheets and programmed keyboards didn't solve the problem of effective entry of narrative data.

2.2. Expanded Search for Appropriate Technology

Because readily available off-the-shelf hardware was inadequate, we decided to investigate hardware and software approaches that could solve the difficult problem of an effective, facile human-computer interface whether or not the technology was commercially available or economically justifiable at that time.

During our expanded search for an appropriate technology, we met Robert Masters and Harlan Fretheim of the Research Division of Control Data Corporation. They jointly developed a cathode ray tube (crt) terminal with a touch-sensitive screen, the Digiscribe, with software to support it. The software consisted of three inter-related programs: the Human Interface Program (HIP) managed user touch-screen selections, the Selection Element Translator (SETRAN) managed the entry of new frames of information into the frame dictionary, and the SHORT operating system managed the execution of programs called from selections.

2.3. Initial Use of a Touch-Screen Terminal

From August of 1967 through February of 1968, we used a Digiscribe terminal connected over phone lines to a Control Data Corporation 160-A computer in Lansing, Michigan, at the St. Lawrence Hospital. Initial versions of the Human Interface Program and the Selection Element Translator were available, and the new software tools required that the programmed keyboard keymats be restructured. This was to be the first of many conversions of the medical content to fit it into an expanded and more capable technology designed to enhance its use. In April of 1968, we installed a Control Data Corporation 1700 computer at Cleveland Metropolitan General Hospital.

The Human Interface Program and the Digiscribe touch-screen terminal allowed medical personnel to be directly interfaced to the computer system. An array of choices was displayed on the screen and users selected one by touching their finger to the screen over the selection. For each of the twenty selection areas, the Digiscribe generated a different character which the Human Interface Program could interpret. Based upon information in the frame, a branch to another frame could take place and new information could be displayed to the user. (See Figure 1.) During normal selection processing, the keyboard was used for special functions such as erasing the last selection, aborting the current message and quickly exiting the system.

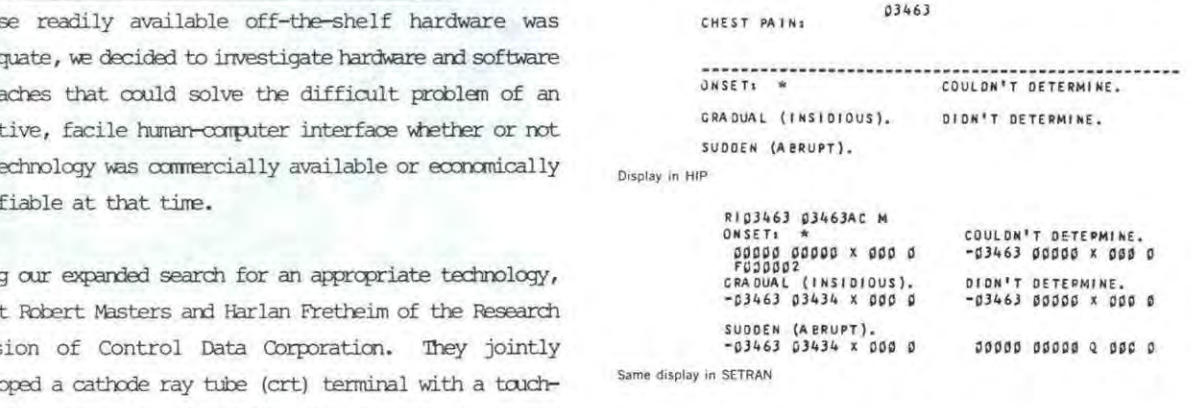


Figure 1. Message generation frame. The "F" parameter associated with "Onset" defines the output format for the title "Onset."

Selection processing used an internal stack of frames waiting to be displayed as well as user written programs that extended the normal selection processing. The result of a user's selections was a message displayed on the top three lines of the screen as well as other information not seen by the user but used by programs to

interpret the selections. This unseen information included the frame number, the choice number within the frame and internal parameters included with the selection on the frame. The internal parameters were used to code selections so that programs interpreted compact codes rather than alpha-numeric data. The internal parameters were the coupling mechanism between the users making selections on the screen and the programs to store, manipulate and retrieve the medical data.

The Selection Element Translator was a program to edit frames. It was used to enter new frames and alter existing ones. The branching among the frames was also defined by the Selection Element Translator. The frames were organized into three types and contained selections that could fit on a single screen. A frame was a physical unit defined by the screen size. The frame types were defined by the function they were to perform, and it was not possible to mix these functions among other frame types. The three functions were: the starting of new messages (and the termination of old ones), the generation of a message and the display of textual information. The first type displayed an array of 7 x 2 choices, the second an array of 6 x 2 choices with the top three lines for the display of the generated message and the third consisted of between 1 and 19 text lines.

We trained physicians and other personnel to use the Selection Element Translator so that the medical knowledge could be entered into the computer without a computer person acting as an intermediary. This ability was fundamental during the development of the system because of the massive number of displays that had to be entered.

The SHORT operating system supported multiple terminals as well as application programs operating in a multi-programming environment. The frames, application programs and all medical data were mass-memory resident. Most selections made by a user required four disk accesses, and the station state data were resident in central memory only while the selections for that station were being processed. SHORT supported four classes of application programs: the first two allowed extensions to the Human Interface Program, one class allowed keyboard input (e.g., the Selection Element Translator), whereas the other did not; the next application program class ran in parallel with the user at the terminal and could communicate with the terminal (e.g., a retrieval program to display information to the crt); the final class executed in the background after the user signed off the terminal. The data generated by the user could be passed to each program class.

The hardware base consisted of a Control Data Corporation 1700 computer and associated peripherals. The CDC 1700 had 32k 16 bit words of central memory with 1.1 micro-second cycle time, and the cpu executed most instructions in 2.2 microseconds. The mass-memory was CDC 854 disk drives, each holding 7.2 million bytes of storage and each with an average seek time of 70 milliseconds. The Digiscribes were connected to the cpu through a display controller with a delay-line for each terminal. The maximum distance between the display controller and a terminal was 1000 feet. The transfer rate between the cpu and the display controller was 50,000 characters per second. The crt screen had 20 lines of 50 characters each. The Digiscribe consisted of transparent, conductive pads on the crt screen and the associated electronics sensed which of the 20 pads were touched. Each pad covered the left or right 25 characters of every other line.



Jan Schultz at a Digiscribe showing a visitor the system.

2.4. Initial medical content development

Developing a system that enabled a physician, nurse or other medical personnel to enter narrative medical information using a touch-screen as the primary interface was a major challenge. Stephen V. Cantrill, M.D., an IBM Fellow during the summer of 1967 (before starting Case Western Reserve Medical School), analyzed a large number of manually recorded Cardio-Vascular problems and defined 90 frames which together could be used to enter most Cardio-Vascular problems. Also, during the same time period, the frames defining present illnesses for GI/Liver

problems were written.

Because of the limitations of using the optical scan sheets for patient questionnaires (which included no branching capabilities, interviewer administration and a high rejection rate), we investigated the feasibility of having the patients use the touch screen for eliciting historical information. Werner Slack, M.D., a medical researcher at the University of Wisconsin using a Linc computer, had demonstrated that directly interfacing the patient with a crt terminal for data collection was feasible for a selected patient population. We questioned whether we could do it routinely for all patients. We developed two short questionnaires for self-administration and found that except for people with poor eyesight, with poor reading skills or with limited ability to read English, the questionnaires could be used easily. Children down to the age of 10 years could also use the touch-screen terminals to answer the history questions.

2.5. Initial electronic record development

The programs to store and retrieve all types of medical data were evolving in parallel with the experimental entry of the narrative medical data using frames. All medical data that we thought could be effectively stored in an electronic record were classified as either narrative or numeric. Narrative data was further classified as resulting directly from selections or resulting from answering questions (translated data). There were three types of data stored in the medical record: direct narrative, translated narrative and numeric data. Keith G. Morgan, a Ph.D student in the Engineering Program at Case Western Reserve University, joined us at Cleveland Metro General Hospital. Morgan developed a program to store numeric data and wrote retrieval programs to display the numeric data in flowsheet form. Cantrill developed a program to translate a questionnaire's responses into narrative text for storage into the patient's record. He also developed a program similar to the Selection Element Translator for the entry and modification of dictionary elements used in the translation process. Lee Stein, a researcher from the original Bolt, Beranek and Newman "Hospital Computer Project" and part of our group in Cleveland, developed a program to transform the user's selections into narrative text for storage into the patient's record. I developed routines to assist in the display of the stored data. One routine, called FORMAT, interpreted codes stored with the narrative data so that the internal form of the record data was output device independent.

The ability of the Human Interface Program to couple narrative selections with internal parameters stored in the frame allowed the frame developer to enter the parameters needed to encode the data and to define the data's output format at the same time the content in the frame was entered.

2.6. Initial phase of development

The initial phase of the development focused on the correct form for the human interface. Other options were tried, but very early on we settled on a crt with a touch screen as the ideal interface for both medical personnel and the patient. At the time we did not fully realize that the interface was effective not only because of the touch-screen, but because of the high data rate of 50,000 characters per second between the cpu and the terminal and the system's ability to process most selections in under .5 seconds.

3. In Search of the Right Development Environment, We Move to Vermont

In July of 1969, a group of five families moved from Cleveland, Ohio, to Burlington, Vermont, and PROMIS Laboratory was formed. One of the original goals for the automation of the POMR was the integration of the patient's bill with the clinical medical record. The hospital in Cleveland was a county hospital, and while we were moving from the initial phase of our research to a phase requiring access to more of the hospital, the hospital's data processing department was shifted to the county data processing department. This meant that there would be no opportunity to integrate the electronic medical record with the patient's bill and the hospital's administrative operation. We began looking for an appropriate site to continue our research.

The University of Vermont's Medical School and Medical Center Hospital offered us the opportunity we were looking for: a teaching hospital closely associated with a University through which our funding could continue and an assurance that the data processing department at the hospital was progressive and willing to work with us.

4. The Medical Knowledge Data Base in the early 1970's

The medical data base consisted of two large bodies of knowledge: knowledge that traditionally resided in medical libraries, journals and textbooks, and knowledge concerning the individual patient. The generic medical knowledge was housed in frames; patient-specific knowledge resided in electronic patient records. This section will describe the frame based generic medical knowledge.

Getting the initial PROMIS system operational on a medical ward in 1970 required a massive amount of medical content to be available on the frames for selection by the medical users. Drs. Larry Weed and his wife Laura B. Weed worked together translating medical content into a form which could be displayed for a medical user. The first step was to create a skeleton structure, as a first approximation, which was a synthesis of remembered experience, a review of classical textbook descriptions of disease, and a review of the appropriate current literature. This process was completed in a year. Once the initial framework was complete, consultants in various subspecialities were used to help continue the building process.

4.1. Drug Information Frames

The drug information frames were constructed by George Nelson, MD., and Genevieve Gilroy, registered pharmacist. The drug information frames provided the medical user using the system with the basic facts about commonly used drugs. The guide for entry of the initial information was focused on the common drugs since most adverse reactions occurred with them. One hundred and thirty of the most commonly used drugs at the Medical Center Hospital of Vermont were compiled, and by 1972 over 180 separate drugs were represented in the system. The information was largely abstracted from standard textbooks and articles in pharmacology, with supplementation from the current literature. The drug data was organized in displays entitled "Check Problem List for," "Side Effects to Watch for," "Drug and Test Interactions," "Usual Dosage," "Mechanism of Action," and "Metabolism and Excretion." All information was reviewed by other physicians interested in clinical therapeutics.

4.2. Diagnostic Process

The initial system was used not only as an aid in making therapeutic decisions, but also in arriving at diagnostic plans. George Nelson did the initial work on the diagnostic plans. This was not diagnosis by computer but used the computer's recall ability and the electronic medical records to assist the physician in the diagnostic process. The first step in the process was the creation of problem formulation sequences for most medical problems and the association of each problem with a unique number, called the problem plan number. Plans were broken up into Get more information and Treatment. The Get More information section was further broken up into For More Data Base, Diagnostic Process, and For Management. The Diagnostic Process was further subdivided into three sections. A single problem plan number could have information about six or seven aspects of the plan.

By August of 1973 there were problem specific diagnostic and therapeutic plans for over 600 problems covering a wide range of medical problems. The plans were audited by outside medical authorities to assure the safety and appropriateness of the decision pathways. Most of the frames were documented by reference citations and the citations were available in the Medical School Library. We did not have enough mass-memory space available to keep the citations on-line. By this time the drug displays had been expanded to include over 300 individual agents.

4.3. Radiology Reporting

The initial system also included frame based medical knowledge for use in support of the radiology department. Peter Dietrich, MD., a radiologist, developed the medical content in this area. The x-ray plans for many problems formulated in the computer were built. A structured reporting format that was both problem and radiologic procedure specific was developed. Reporting sequences for over 150 problem/procedure pairs were built by October of 1972.

4.4. Problem Plan Numbers and Tables

In order to accommodate the problem specific information in the system, a new container for medical knowledge was necessary. The information in the container was never seen directly by the user, but was needed to collect together all the frames that related to one problem plan number. The new container was called the branching information table and it had slots for each type of problem specific information. Each slot contained a frame number that could be put into the branch stack that the Human Interface Program used, so that if a specific selection were made on a frame and the patient's problem were known, then the problem plan number could be used to look up the correct branching information table slot

for the selected problem. The branching information table was the first of many tables that would come to exist as part of the medical knowledge data base.

The table is the container for all machine-readable data related to a medically defined problem or procedure or other entity such as a system user or terminal. Each table has a unique address or code. Most of the data in a table is not visible to the medical user, but is used by programs to perform actions or do special display branching.

5. An Operational PROMIS on Medical and Gynecological Wards

The PROMIS system replaced paper records with a problem-oriented electronic computer record for six months on a general medical ward and nearly four years on a gynecology ward in the Medical Center Hospital of Vermont. This development took place from July of 1970 through November of 1974. More than 3000 patients were cared for using this system during the time it was in operation. More than 500 individuals (physicians, nurses, medical students, house staff, radiologists, pharmacists, social workers, ward secretaries, dieticians and laboratory technicians) used PROMIS as the sole mechanism to add and retrieve information in patient records. The prototype PROMIS system demonstrated solutions to many general problems seen in other computer applications. Through redundant hardware, careful maintenance, and good diagnostic tools, the system was made available to users more than 99.6 percent of the time, within a scheduled 24-hour, seven-day week operation. The system was designed for fast response to facilitate use, and would respond to a user's selection by presenting another screen within 600 milliseconds for over 50 percent of the selections. Response was both rapid and comprehensive as patients' complete records were available on the terminal during their hospital stay as well as more than 30,000 displays of medical information available for guidance.

Eligible personnel throughout the institution were allowed access to the electronic record. Terminals on the ward, in the operating suite, in the pharmacy, in the clinical laboratory and in the department of radiology demonstrated the synergism possible using an electronic record.

The overall goal during this time period was getting PROMIS operational on a single hospital ward and analyzing the operational hardware and software system in enough detail to allow realistic estimates for the size, cost and system structure necessary to implement PROMIS throughout a hospital.

5.1. The Software to support an operational PROMIS

The software developed to support the operational PROMIS included the SHORT operating system, the Human Interface Program and the Selection Element Translator, supplied by CDC. The Store program transformed the data generated by the Human Interface Program into patient record and other system files. The Retrieve program abstracted information from the patient's record or other system files and displayed the abstracted data in the form of a narrative report or a flowsheet. When all of the application programs were written for the CDC 1700 system, there were over 250,000 lines of assembly language code. (Schultz, Cantrill, Morgan 1971)

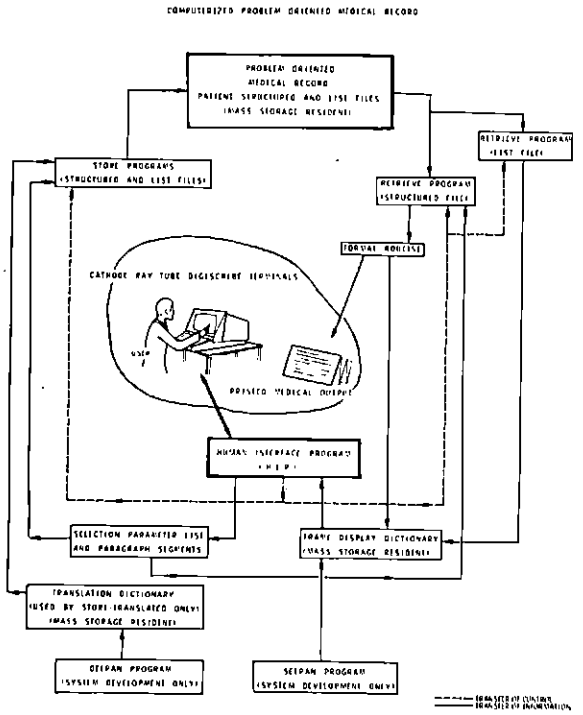


Figure 2. The Elements of the Application Software.

5.2. The Form of the Data Within the Electronic Record

An individual patient's electronic record was mass-memory resident and required a structure that facilitated its manipulation while minimizing the number of mass-memory accesses required to perform the manipulation. An individual record consisted of one fixed length index block and a variable number of fixed length data blocks. The index

block was an index to all the data in the data blocks and was accessible using the patient's system identification number. The index block contained a variable length list of pointers to the data blocks along with other information to identify the type of data in the data block. To access data in the patient's record, a search was performed through the index's data block pointers looking for those pointers that satisfied the retrieval request. For any single data item retrieval, two mass-memory accesses were required.

The internal form for a data item within the electronic record depended upon whether the item was narrative or numeric. For narrative data, the internal parameters associated with the selections were interpreted as format codes to be used to define the retrieval format. For numeric data, the internal parameters associated with the selections were interpreted as data types and each data type was stored in a separate fixed length field. Consider the following example: a temperature's internal parameters were an internal code number, a time (10:23), a date (February 23, 1970), a title (Temperature), a numeric value (38) and a numeric qualifier (degrees Centigrade).

5.3. The Selectable List Files

To facilitate access to rapidly changing patient record information, it was necessary to present the patient information on displays for selection. A list of patients on a ward or the list of current medications for a patient are examples. Displaying the lists for selection required that the data to be displayed be readily available so that it could be retrieved responsively. The selectable list files were an abstraction of data stored in the patient's record to facilitate the rapid retrieval of selectable lists.

5.4. The Medical Guidance To Support An Operational PROMIS When health care providers sat down to use the terminal, they first selected from a list of patients presented on their terminal screen the name of the patient with whose record they were to interact. Having chosen a patient, the providers selected one of the sections of the medical record - the Data Base, Problem Formulation, Plans and Progress Notes - in order to add to or retrieve from that part of the patient's record.

The Data Base

The nurse entered the patient's chief complaint, patient profile and the general appearance and vital sign components of the physical examination. The patients, after only two-to-four minutes of instruction, entered their own review-of-systems by answering a series of up to 275 yes/no and multiple choice questions. Only the positive responses were stored in the record. The Review-of-Systems was then retrieved and reviewed by medical students and house officers. Positive responses were developed into present illnesses by touching selections specific for each symptom. The physical examination would then be entered using up to 1410 frames, the number actually used being a variable related to the patient's physical status.

Problem Formulation

After completion of the Data Base, the physician formulated the patient's problems by making a series of selections on displays. Each problem would be stated at its level of definition, ranging from a symptom to a firm diagnosis. The displays facilitated further description of problems by guiding the user - frequently providing them with criteria to help use consistently precise language. These criteria or definitions were taken from the literature with a library reference citation noted on the display. The system facilitated easy updating of any problem when new information gave added insight. Problems could be added or their status changed (e.g. from active to inactive) at any time.

Plans

Each component of the plan was individualized for many problems in the system. The system contained displays based on the current literature detailing how to work-up, treat and follow the problem. These displays removed reliance on memory and allowed a new intern to work-up the problem with a degree of accuracy and completeness comparable to a subspecialist. The displays did not constrain the users; they were free to order whatever they pleased - but the displays served to guide them. More than 600 specific disease entities had some degree of problem-specific information in the system.

Progress Notes

All notes written about any problem after the Initial Plans for a problem are Progress Notes. These included physicians' notes, nurses' notes, and notes of social workers, chaplains, and all who had a recorded contribution to the patient's care. All Progress Notes were entered by first selecting the appropriate problem on the terminal

screen from the patient's list of problems. The Symptomatic and Objective sections took the user to frames specific for the body system of the problem. The nurses were encouraged to enter Progress Notes according to the severity and rate of change of each problem of each patient under their care. They used the same displays as the physicians, although some displays were used more by nurses than by physicians. Operative notes, radiology reports, clinical laboratory reports, social service consultations, and specialists consultations were also entered as Progress Notes under the problem they addressed.
In September of 1974 there were approximately 30,000 frames in the system. A breakdown showed that 6000 of these were Data Base frames; 4000 were Problem List frames; 7000 were Plans frames; 8000 were Progress Notes frames; Superstructure and others made up the remaining 5000 frames.

5.5. Data Retrieval

The above four sections comprise the basic patient record. This data was kept in electronic form in the computer system during a patient's hospitalization. There were no written records maintained in parallel. The updates to the record were printed each day and were available on the ward. Patient information could be retrieved at any time at any terminal by an appropriately identified medical user. Since the record was in electronic form, data could be retrieved in many ways. Information could be retrieved on a specific problem, on all problems chronologically, or in flowsheet format. Current outstanding drug orders, investigations and accumulated charges were also available by problem. The problem of absent or lost records was non-existent. The records were all legible. Any number of users could simultaneously retrieve from or add to a record.

5.6. Problem Oriented List of Charges

One of our initial goals, the creation of a Problem Oriented List of Charges was accomplished. We compared a hospital bill with a list of charges calculated from the chargeable items in the electronic record. We also presented a patient's problem list with the charges associated with each problem so that medical users and administrators could see the cost of treating each problem. We discovered that the Problem Oriented List of Charges created from the electronic record was different from the hospital's bill since it represented everything that was recorded in the record and not just those things that had a charge slip written for them.

5.7. The Hardware to Support an Operational PROMIS

The hardware base consisted of two Control Data Corporation 1700 computers and associated peripherals. Each central processing unit had 32k words of central memory and had seven disk drives with 7.2 million bytes capacity each. There were 22 local terminals and 8 remote terminals. Remote terminals operated at 4800 bits per second and contained their own delay line refresh memory. The remote terminal multiplexor communicated full duplex at 4800 bits per second. A modem at each end of a C2 conditioned, leased 4-wire phone line was used to connect the remote terminals to the terminal multiplexor. The 4800 bps modems cost over $10,000 each and had to be manually equalized to the telephone line.



This is 50 megabytes of mass-storage in 1970.



Ernie Preiss working on the CDC 1700.

5.8. Information Utility

The continuous operation of the system required the development of a reliable, backed-up hardware base that the users considered to be an information utility. One

of the major elements that contributed to keeping the system in continuous operation was our ability to see patterns in the occurrence of various system problems. Whenever a problem occurred that affected the operation of the system or more information on the problem was available, we described the problem in problem-oriented fashion and posted it so that periodically we could discuss the current operational problems. We kept paper logs of all operational problems since there was not enough mass-memory space available to keep the data on-line. Eventually, when more mass-memory was available, we kept the problem statements on-line. There were eleven major problem categories, divided into hardware, software, architectural relationships and undefined. Each category was divided into maintenance, operation, environment, financing, personnel, development and education. Problems in the hardware, central processor and peripherals operation category ranged from "core parity errors" to "idle time low for unknown reasons." Problems in the hardware, environment category included "Ernie's tools disappearing" and "cpu room floor dirty and place a mess." The total log for this time period included over 500 active problems. One of our mottos for the period was to "let the problem lead us," and we had enough problems; the only question was where it was to lead us and which problem would be the leader.

6. The National Library of Displays

The medical group within PROMIS Laboratory was formed to create and maintain the thousands of computer displays. The medical content was viewed as a textbook or encyclopedia organized and available according to the patient's problems. As Dr. Weed wrote in 1972:

> "The opportunity now exists in modern medicine for making available to health care personnel tools which will permit them to perform with excellence without depending on an encyclopedic memory.... It is therefore an appropriate goal to seek to make available to all health care personnel a tool which amplifies to a uniformly excellent level their initiative, memory, and ability to create and to execute reasoned and disciplined plans.... The goal can be accomplished in large part if the tool has built into it the parameters of guidance and currency of information required by health care personnel as they perform and record their work. Then, individual human memory and initiative will cease to be the critical link between all we know and all that must be done."

6.1. The Process of Medical Content Development

The process of creating the medical content involved much more than just entering the medical knowledge onto the frames. Prior to the release of the medical content for on-line use, a rigorous in-house audit was completed. Outside experts in the relevant medical subspecialties were invited to Burlington for a 3-5 day period, during which time they audited each frame sequence for safety, accuracy, currency and completeness. The audit comments were then reviewed by the author of the frame sequence and appropriate changes were made. Each step of the process was documented. Finally a check of the branching was done to verify that the sequence worked as planned.

6.2. Preliminary Study by National Bureau of Standards

The process outlined above was limited by the resources of the medical group of PROMIS Laboratory. It became clear that prior to the widespread distribution of systems with the characteristics of PROMIS, a national body should be created to assume the responsibility for the review and distribution process. This would insure that the systems represented accepted standards in medical care as well as computer performance. The National Bureau of Standards in conjunction with the Health Services and Mental Health Administration sponsored a study directed at identifying actions appropriate for facilitating the wide spread use and maintenance of PROMIS. Melvin Conway, PhD., was hired by the National Bureau of Standards to do a preliminary study.

The report from the study was available in October of 1972 and was an initial planning document. It raised many issues: display library accreditation, standardization, distribution, quality control, the administrative tools required by the library, certification in the field of the record-maintenance systems which are using the displays, and controlled evolution of the entire process.

The report defined terminology and concepts which we incorporated into our thinking and ultimately into our evolving system. It defined Generic system specifications which set forth criteria for initial certification for record maintenance systems so that system vendors could design to the specification. It defined under this a system line which is a definition of a class of concrete record maintenance systems and under this a system instance which is an individual medical record maintenance system.

A parallel three level hierarchy of definitions existed for the collection of displays. The generic library is the primary standard resident in the national repository from which all specific display collections residing in system instances are drawn. A sub-library is a subset of the generic library to allow distribution of non-identical display libraries. A library instance is a machine-readable data set of a sub-library intended to reside in one system instance. And finally, a library instance update is a portion of a library instance which is sent out to a system instance when only part of its display collection needs to be changed.

6.3. Release of Frame and Table Library

PROMIS laboratory was committed to seeing a National Library of Displays formed. We resisted efforts by the government and private industry to get early releases of the display library; we feared if multiple copies were released, it would become impossible to provide updates since the mechanisms for creating sublibraries and library instances had yet to be worked out. The merging of already released library instances with a library instance update was a difficult problem. The tools to perform the library instance update were not in place until 1979.

Our National Center for Health Services Research contract in 1976 required that a data description language be developed to externalize the frame and table library in a form that was machine independent. We made available on magnetic tape yearly the complete library as well as a copy of all of the computer software from 1976 until the last contract expired in September of 1981.

7. Expandable and Exportable Architecture Is Defined

Besides the continued operation of PROMIS in the hospital and the entry of new medical knowledge, a major effort was undertaken starting in the spring of 1973 to define an architecture that could serve a full hospital as well as be exported to other medical settings and different sized institutions. Wesley Clark and Mel Conway were hired by the National Center for Health Services Research to assist us in this task. The operational CDC 1700 served as a resource for defining the subsystems for a new architecture. The CDC 1700 system consisted of three subsystems: The Human Interface Manager had both hardware and software elements. The station hardware included the cathode ray tube terminal, the Digiscribe unit to make the screen touch sensitive, the hardware logic to make the terminal behave in the expected way and the display controller or multiplexor that interfaced the terminal and the cpu. The software to control the human interface interpreted the frames, concatenated the selections together to generate English narrative, determined the next frame and presented it to the user for selection and then re-initiated the cycle. The Patient Record Manager included the mass-memory to store the patient's record as well as the software to update, retrieve and take patient data on and off-line. The Frame Manager included the mass-memory to store the frames and tables, and the software to create and modify them.

7.1. Subsystems Characteristics

Each of the subsystems had characteristics which were used to helped us determine the means of instrumentation. These characteristics were determined by the way the system was implemented and by the way it was used within the hospital. The major characteristics of each subsystem are described below.

The Human Interface Manager

This subsystem had to be extremely responsive. We defined extremely responsive to mean that the time from user selection to complete display of the next screen's data was less than .25 second, 70% of the time. This responsiveness was necessary to facilitate the effective interface of professional non-computer trained people to a functionally complex computer system. The human engineering of the touch-screen is essential for the interface to work well. The data rate between the frame manager, the human interface manager and the station must be high. Only 50 milliseconds was allocated for the transfer time between the frame manager and the station. The displays used to generate data were sparse, with an average of six choices used out of a maximum of 14. The combination of the highly responsive system with sparse displays facilitates pattern recognition and rapid interaction by the user. Unlike other computer systems where the function of the human interface was to get to a computation, in our case the interface was the computation.

The Patient Record Manager

This subsystem must be extremely reliable so that patient data is never lost and so the many types of retrieval requests for individual patient records as well as for groups of records are handled responsively. Individual

records ranged in size from 10,000 to 100,000 characters (or more) with an average of 6,000 characters of data added daily. The access time to one element of a patient's record was 65 milliseconds, and it took from 2 to 200 accesses for most retrievals. An average storage into a single patient record required 6 accesses to the record file and 20 to 100 accesses to the input data generated by the Human Interface Program.

The Frame Manager

This subsystem contained 30,000 frames and potentially could contain two to three times that number. Over 18 million characters of storage were required, and the average number of characters per frame was 600. The 600 characters included not only data to be displayed to the user, but other branching and internal parameters. The internal form was not densely packaged; groups of fixed length fields were present whether filled or not. Two mass memory accesses were required to retrieve one frame. Twenty percent of all accesses were to two percent of the frames, so that a faster access time storage media for a small number of frames could have an impact on the through-put of this subsystem.

7.2. Potential Expansion of the CDC 1700 System

The CDC 1700 system could not be expanded to operate the total hospital, not only because it was technologically disadvantaged, but because the system software had many limitations that could only be changed with a major software restructuring. It is valuable to consider the types of walls present in that system. They became the guide to a redevelopment effort. Each wall was a design decision in the Human Interface Program or the patient record structure. The Human Interface Program's design restricted the maximum number of selections that could be made within one generated unit so that frame sequences had to be changed to keep the generated unit from overflowing; the frame library could contain a maximum of only 32,000 frames, and we were very close to overflowing it. The structure of the patient record restricted the size of the patient record index to a single mass-memory block so that some very long patient stays would overflow the block requiring the starting of a new record; the numeric information for one data item in the record had a fixed maximum size; a maximum of 144 patient records could be on-line at one time; the patient's problem and order lists had a maximum size so they had to be monitored and cleaned up as some of them reached maximum size; the numeric and narrative data were stored in separate parts of the record and retrieval functions that operated on narrative data couldn't be applied to numeric data and conversely. This made it impossible to retrieve a flowsheet of narrative data.

7.3. Incorporate Expanded Functions

It was also necessary to incorporate expanded functions into the system's operation. We wanted to be able to fully exploit the electronic record: by allowing its full access in all the supporting areas of the hospital; by performing data compression of the electronic record; by making available at all times the patient's problem list and current medications for all patients in our population; by developing a patient record that could span multiple admissions and serve for the patient's lifetime; and finally, by linking the medical data in the record with the patient's financial data.

7.4. Scale to Support Total Hospital

The new system had to be able to support the total hospital. The size of the system was determined by the number of active patients in the system at one time and the total served by the hospital, by the size of an individual patient's record, and by the number of terminals active at one time.

The Medical Center Hospital of Vermont in 1973 handled 20,000 admissions and 120,000 out-patient visits per year. There were between 25 and 85 admissions per day and 10,000 out-patient visits per month.

The size of a single patient admission was determined by looking at a "typical" six-day period of system operation and extrapolating. We used figures for the number of admissions per year and out-patient visits per year to arrive at the amount of patient record growth expected in one year. It was 1.24 billion characters or the equivalent of 4 double density IBM 3330s (the measuring stick of that era).

The number of terminals for the hospital was calculated based upon 4 terminals per 20 bed ward and 7 terminals per 34 bed ward and terminals in all the supporting areas. The total number was 200.

7.5. Architectural Characteristics are Defined

Given the characteristics of the functional structure of

the CDC 1700 system and the characteristics of the site for installation, an architecture for a PROMIS system was defined. The major elements of the PROMIS architecture were redundancy of hardware elements for reliable service, guaranteed responsiveness with minimal sensitivity to load, and access to any patient's record from any terminal within the system. The architecture had to support the locational diversity of the health care system since health care is not practiced in one geographical site, yet it was important to allow communication among the sites.

7.6. Two Remaining Issues

Besides the elements for the architecture listed above, there were two issues that had to be explored before a final decision on an architecture could be made. The issues were: should the subsystems be functionally partitioned into separate hardware elements, and should the patient record files be centralized or distributed?

One potential architecture was a network of mini-computers. Each node of the network was to be one mini-computer that would handle between 10 and 30 terminals. The number of terminals per node would be determined by requiring a system response of less than .25 second 70 percent of the time. The nodes would each contain the three subsystems. A communications medium would allow the nodes in the network to communicate. The patient records which were normally accessed by the terminals connected to one node would be contained in that node. Other nodes' access to a patient record would be on-demand and would require a transfer across the network. For storage into a record, packets of data would be shipped to the node which contained the patient's record.

Another proposed architecture was a functional partitioning of each subsystem into separate hardware elements. There would be Frame Engines containing the frames in a read-only form. Each Frame Engine would be a small mini-computer with mass-memory to contain the frame library and communication ports so that the Human Interface subsystems could communicate with it. The Human Interface subsystems would contain no mass-memory and would multiplex many terminals. Each Human Interface subsystem would have communication ports for connections to both a Frame Engine as well as a Record Processor. The Record Processor would be fully duplexed mini-computers with communication ports for connection to the Human Interface subsystems and to remotely located printers.

The issue of whether the patient record file should be centralized or distributed was posed in the following way: can we build a system that can serve New York City as easily as it serves the City of Burlington and the State of Vermont? Ultimately there would have to be a medical record network serving a very large region; the question was, shall we learn the problems of building such a network while it was small or wait and solve the problems posed by such a network only when it was no longer feasible to keep the data files centralized?

A paper simulation was done of the system data traffic among the subsystems, assuming the architectural model of a network of six mini-computers with each mini-computer managing 50 terminals and with intra-node traffic for a distributed patient record file being 25% of all accesses. The average total data rate was 42,000 characters per second; assuming a burst rate of ten times that rate and assuming maximum intra-node traffic for the patient record files, the data flow would require only 25% of a million word per second I/O bus. The remaining bus capacity would be available for instruction execution, data pool swapping and program loading. Given the system data traffic requirements and the type of hardware available at the time, there was no need to segment the functions into separate hardware units.

7.7. Final Decision

Our decision was to define an architecture consisting of a network of mini-computer nodes with each node containing the three subsystems. The patient record files would be distributed across the network with the records for the patients served by the terminals connected to the node defining where in the network the patient records would be located.

The architecture supported a distributed patient record file since ultimately not all records could be kept in one centralized system and we wanted experience handling a distributed patient record file while the files were of reasonable size. We wanted to solve this problem by design, not by being forced to consider it when the centralized system could no longer expand.

8. Specify Technology to Support the New Architecture

With the architecture defined by the end of 1973, our next year's jobs were set out before us. A mini-computer, terminal, inter- and intra-node communications bus as well as a programming language had to be specified and selected for the new PROMIS system. We had been looking for a hardware person to round out the PROMIS computer group when James Wanner walked through the door. An astronomer and mechanical engineer with digital electronics experience, his first task was the writing of a terminal specification. We decided to use MOS technology where it could be applied. MOS was "mostly off the shelf," and those items we could not find on the shelf we tried to put on the shelf. Detailed specifications were written for all of the items and we set things in motion to begin the procurement process.

8.1. Mini-computer specifications

The mini-computer had the following specifications: central memory of at least 128k, 16 bit or greater words with a cycle time of one microsecond or less, and direct memory access to peripherals; the environment protection had to include power failure restart circuitry and error checking on all data paths in the system; multi-level interrupts; memory protection consistent with any memory mapping or multi-programming executive; mean time between failure of less than one processor failure per year. Mass-memory was divided into 10 millisecond fast access for 2 million characters and 65 millisecond access for 200 million characters. Communications subsystem (four wire or phone) must support printers and inter-computer communications at 50,000 bits per second or faster. Up to fifty local touch screen terminals operating at 16,000 characters per second or faster had to be able to be connected to the system. Other standard peripherals had to be available.

8.2. The Touch-Screen Specifications

The touch-screen terminal had the following specifications: 8 1/2 inches high by 11 inches wide screen; two display modes - the first - 24 lines by 80 characters each, with 12 lines aligned under the touch-pads on the screen, and the second - 36 lines by 100 characters each with the bottom line aligned with the bottom line of touch-pads on the screen; alphabetic characters shall appear in at least three distinct ways; if a cathode ray tube is used, the display shall be flicker-free for all users; the touch pad screen shall have an array of transparent touch pads each covering a fixed block of characters in a displayed page of text. The pads shall respond to the operator's finger touch, independent of body position. When a pad is touched, the text lying under the pad shall be emphasized. This emphasis will continue until a new screen-full of text is received by the terminal for display. The bottom line of screen display text shall have 10 touch pads, each pad covering 7 characters, with a one-character-wide space between the pads. Eleven additional lines of text, on approximately 5/8" centers, shall have four touch-pads each covering 19 characters of text. Parallax errors between text and touch-pad positions shall be minimized by mounting the pads as close as possible to the text. Data flow into each terminal shall be at the rate of 15,000 characters per second, or faster. Data flow from each terminal shall proceed at a rate of at least 15,000 characters per second, or less if appropriate to the terminal controller subsystem. The terminal interface subsystem shall support as least 24 terminals in apparent concurrent active use. Each terminal shall have access to more than one central processor unit, for use as back-up on cpu failure. Portable terminals shall be provided by means of wheeled cart mounted terminals with single plug interface to the system and taps at various locations.

8.3. The Communications Subsystem Specifications

The communications subsystem had to be able to support up to 50 touch screen terminals operating at 15,000 characters per second full duplex, up to 20 printers operating at 9600 characters per second and at least one inter-computer connection operating at 50,000 bits per second.

8.4. Programming Language Specification

The programming language specification included the following description. A high level language is desirable, but not at the expense of execution speed or mass-memory access efficiency. The language includes character manipulation instructions. It is machine independent so that the application programs do not have to be rewritten when the machine is changed and so that the same programs can run on multiple machines. The language must be transparent to the operating system. The language must incorporate the ideas of structured programming. There can be separate languages for application programming and systems programming. Considering the characteristics of the original assembly language programs: very efficient, very difficult to change, very little documentation as part of the program; do better this time around.

9. The Architectural Elements

9.1. The Mini-computer Procurement

Request for proposals was sent to eight computer manufacturers. Four "no bid" the proposal. IBM said they had a "hole" in their product line and that a network of their computers would be too expensive. Digital recommended an 11/45, which met the hardware requirements, but would not make information available on their software. Although they offered to sell us hardware, they made no bid on our request for proposal. PRIME and INTERDATA made no bid.

We received four proposals. The Modcomp, Inc., proposal met the hardware specifications, but the system they proposed, the ModComp IV, was not yet available for testing. Control Data Corporation sent an extensive proposal; William Norris, the CDC President, delivered it and spent the day at PROMIS Laboratory discussing the continuation of our long term relationship. CDC proposed the most interesting system. It was a system composed of 10 peripheral processors from a CDC 7600 with a large shared central memory. However, the system was not a standard product, and a simulation showed that for our type of architecture the multi-processor system got in the way of responsiveness. The Data General proposal met the hardware specifications, but the software could only address 96k of the 128k address space so it did not fit the software specifications. Edson DeCastro, President of Data General, visited PROMIS Laboratory to discuss their proposal.

Varian Data Machines submitted the winning proposal with their V77-400 computer system. It fit both the hardware and software specifications and the company was interested in working with us on the joint development of a communications subsystem to support the high-speed touch screen terminals.

9.2. The Touch-screen Terminal Procurement

There were three departures from the main stream in our terminal specification: the touch-screen interface; the dual display modes: 24 lines by 80 characters for selection and 36 lines by 100 characters for text retrieval; and the high speed data transfer. After discussions with over 30 potential suppliers, we decided to drop the dual display modes as an essential requirement. We received eight responses to our specifications, and MEGADATA Corporation was selected to build the new high-speed, touch-screen terminals. The terminal was a random-logic PDP 8 so that it could be programmed to perform our specialized functions. It had expansion slots for the two special boards to be built by MEGADATA, one for the touch screen and the other for the high-speed line interface. The touch screen did not have set pad locations on the screen, but was an x - y digitalizer that had a resolution of .25 inch. It sensed a finger touch via echoing of surface waves. The first group of 20 terminals cost $8,500 each.

9.3. Communication System Procurement

The terminal-to-computer communications remained largely undefined until a computer system was selected in early March, 1974. Soon thereafter a suggested communications system was formulated, using the standard Varian Data Communications Multiplexor and VTAM software as a starting point. A research contract was signed with MITRE Corporation in July, 1974, to assist us in designing and implementing a communications system which would support digital traffic on a CATV-type distribution system for the PROMIS architecture. The resultant system was to provide highly reliable, low error rate and flexible reconfiguration capabilities for the PROMIS digital signals among the many terminals and several computer nodes. The data rate was to be at least 15,000 characters per second. Of secondary importance was the ability of the system to simultaneously support the communication of other services such as entertainment and educational television. MITRE was selected for this work because of their pioneering efforts with the MITRIX system which used CATV technology to build a time-division multiplexing scheme on a CATV bus for inter-node data transfer. They had developed CATV modems as part of the MITRIX system.

9.4. Programming Language Development

The only high level language available on the V77-400 was FORTRAN. Its code was not re-entrant, and it had an extensive run-time environment that was not appropriate for our tasks. We had done extensive programming language research. We wanted a language with syntax that would support structured programming, a semantics that would support the manipulation of logical records of variable type and length and a pragmatics that would support frames, records and indexes to both. We wanted to be able to manipulate strings that could grow to be very long (up to 32k characters) and to be able to access mass-memory resident data logically. Facile control of the touch-screen terminal was also a requirement, as was

network access to the logical data structure and network access to multiple communicating processes.

I designed a programming language based upon our CDC 1700 assembly language experience and a study of other languages. Morgan and I wrote an interpreted version of the language on the CDC 1700 with semantics as defined above. The interpreted version accessed the data using a structure similar to a list processing languages' "property lists." Extensive searching through the data structures was required for any operand access, and consequently it executed very slowly. It would "run like the wind" and could take more than 30 seconds to process one selection.

We decided to develop a compiler language, the PROMIS Programming Language (PPL). PPL was a combination of a high-level procedural language with a very powerful embedded data base management system. It included procedures to manipulate the touch screen, schedule and sprout processes throughout a network, transfer data across the network, and manipulate strings effectively. It was efficient in terms of CPU cycles because it was not an interpreter; all data access to 16 bit numerics was in-line code and the internal form of the data required no searching. Each data element was accessed either with an absolute address or with a pointer variable and a relative address which was bound at run-time. Cantrill wrote the code generator, and I wrote the scanner and parser for the compiler. It compiled source code at 2,000 - 6,000 lines per minute.

The syntax of PPL was a major departure from "ALGOL-like" languages of that era. PPL's assignment operation was from left to right, the way one reads English. PPL had no explicit statement delimiters, that is, no ";" after each statement. PPL's comments could appear after implicit statement delimiters, and except for an * at the start of a new line, there were no explicit comment delimiters. Control statements all had explicit statement terminators. For example, an IF statement was terminated by an ENDIF or FI. Lists of statements could replace a single statement without requiring a BEGIN and END for the block. The syntax of a control statement required it to be spread across multiple lines, and no more than one statement could be put on a single source line.
The PPL syntax supported our ideas of what "structured" code should look like.

PPL was designed for the programming of applications. We could not afford to wait until the compiler was done before we began programming its run-time support routines. We were also concerned about the efficiency of PPL for systems programming tasks; since PPL code was to be machine independent, we were also concerned about distorting PPL to fit various systems programming tasks. For these reasons, Cantrill developed a structured preprocessor to the Varian supplied macro-assembler. Called STRAP, it allowed all assembly language code to pass through. Statements recognized as STRAP commands caused output of the appropriate macro-assembler instructions. STRAP was originally written in STAGE2, a machine independent macro-processor, but it processed code too slowly. STRAP was then written in itself and could process up to 5000 lines of code per minute.

10. Software and Hardware Specification

Our funding agency was devoting a large percentage of their budget to our development efforts and wanted to be assured that we were up to the task. An advisory committee was put together to oversee our development. The committee included Ivan Southerland, Allen Newell, George Robertson, Herbert Sherman and Jack Hall, M.D. Newell and Robertson had previously developed a menu selection system called ZOG, which sensitized them to our work. They applied the PROMIS interface ideas of a rapid response, large network system to ZOG.

The implementation of the new architecture involved coordinating efforts of multiple hardware vendors and the PROMIS Laboratory computer group. We had a limited budget and limited time. We followed D. L. Parnas's advice. The implementation used "specifications sufficiently precise and complete that other pieces of software can be written to interact with the piece specified without additional information... . The specification must be sufficiently formal that it can conceivably be machine tested for consistency [and] completeness [in the sense of defining the outcome of all possible uses]... By this requirement we intend to rule out all natural language specification." (Parnas 1972) We had been using decision tables for program development for a few years. Henry Beitz, a CDC consultant, convinced us that an unlimited entry decision table could be used as the basis for demonstrating an understanding of a problem and also as the fundamental design document. Decision tables were used to design and test the interface among hardware and software modules.

The run-time environment developed to support PPL (described below) was defined in 17 decision tables and 78 rules. The definition of the polled multidrop protocol to connect the high-speed terminals to the CPU was a set of 12 decision tables with a total of 77 rules. The protocol was used by Varian Data Machines, by MEGADATA, and by PROMIS Laboratory. When the hardware was delivered and the terminals were connected, the system operated correctly the first time it was tried. One problem was discovered in an error pathway in the hardware, and six problems requiring software modification were discovered. We also used "thin-wire" protocols to define our run-time environment. We sent messages among three separate tasks: the PPL code environment to manage the resources of the cpu, Terminal Input/Output to manage the terminal and printers, and Block Input/Output to manage mass-memory.

11. Run-time environment supporting PPL

The PPL run-time environment consisted of facilities to support the work requirement modularized into processes to, support the internal form of the PPL logical data type - the paragraph, and to minimize the number of mass-memory accesses.

11.1. Process Support

A process required access to three types of resources: the cpu for execution of PPL code, the mass-memory devices for access to the PPL paragraphs, and the terminals and printers. Each of these three types of resources was accessed using a task in the Varian Data Machines VORTEX II operating system. Based upon the resource required by the PPL code being executed, messages dispatched the PPL process among the different tasks.

11.2. Support for the PPL logical data type

Files and individual blocks were manipulated by the file system. The PPL logical data type was broken up into paragraphs, sentences and sentence elements and were manipulated by Sentence Input/Output routines. The file system, developed by Henry Stambler, used the VORTEX II software as a foundation and provided the following enhancements:

1) A file was configured as a set of discontiguous components spanning several VORTEX file areas or disk packs. Up to 64 components were allowed for each file.

2) New components could be added as needed.

3) A file could be as large as 2048K blocks. In VORTEX the maximum was 32K sectors. A block could be up to 2K bytes long.

4) The table defining the files resided in central memory so that any block could be obtained in a single disk access.

5) Software checksumming and read-after-write validation were provided.

The Sentence I/O routines, developed by Morgan, performed the mapping between the PPL operands and words within the file system blocks by maintaining state variables and a map-in area for each environment, making possible all the logical and string operations in PPL.

11.3. Mass Memory Access

PPL programs required rapid access to large amounts of data. It was impossible to keep even a small fraction of this data in central memory, and it was undesirable to access mass memory every time a piece of data was needed. A memory buffer pool, developed by James LeMay, housed all data accessed via PPL instructions. The data in the pool ages, and because a PPL program normally accessed the same information many times during its execution, the pool frequently still contained the information. The pool was a resource used by the resource management tasks; it was transparent to the PPL programmer. The ratio of block to disk accesses was typically between 2:1 and 25:1, based upon the type of program and the amount of data it needed.

12. The Library System: Management of the Medical Knowledge

Medical knowledge entry was controlled in a manner similar to patient record data entry and retrieval. The library builder makes selections at a touch-screen terminal. By answering questions and selecting the next step in a series of actions, the builder was guided through the steps required to add new information to the data base or update previously entered knowledge. The library system was developed to manage the frame and table library, in the same way that PROMIS was developed to manage the patient care functions. PPL was used to implement the library system on the same hardware base that the PROMIS system was implemented.

The library system managed frames and tables.

12.1 Tables

Tables were developed for many uses. The information in

a procedure table is used, for example, to:

1) determine branching to a subroutine of frames at certain points in the frame sequence;
2) define information that is specific to a site (such as drug prices and inventory levels);
3) define lists of logical actions that should take place when this procedure is stored in a patient's record;
4) specify lists of synonyms for the procedure to make it accessible on various alphabetic lists;
5) classify the procedure for all uses;
6) point to other data structures that are related to this procedure;
7) list all frames and tables that access this table.

12.2. Frames

The frames are interrelated in a network; each choice on each frame points to a frame to be displayed when that choice is selected by a user. The network is a guidance system in which every frame is simultaneously medical content and structure for the next finer level of content. The most general frames, superstructure frames, serve as high-level indices to other frames in the network, establishing the context (such as the current section of the medical record) the user will be working in. As the user progresses through the network, the frames become more specific, containing, for example, drug information or lists of diagnostic procedures, and enabling procedure ordering and reporting. (See Figure 3 for a frame sequence)

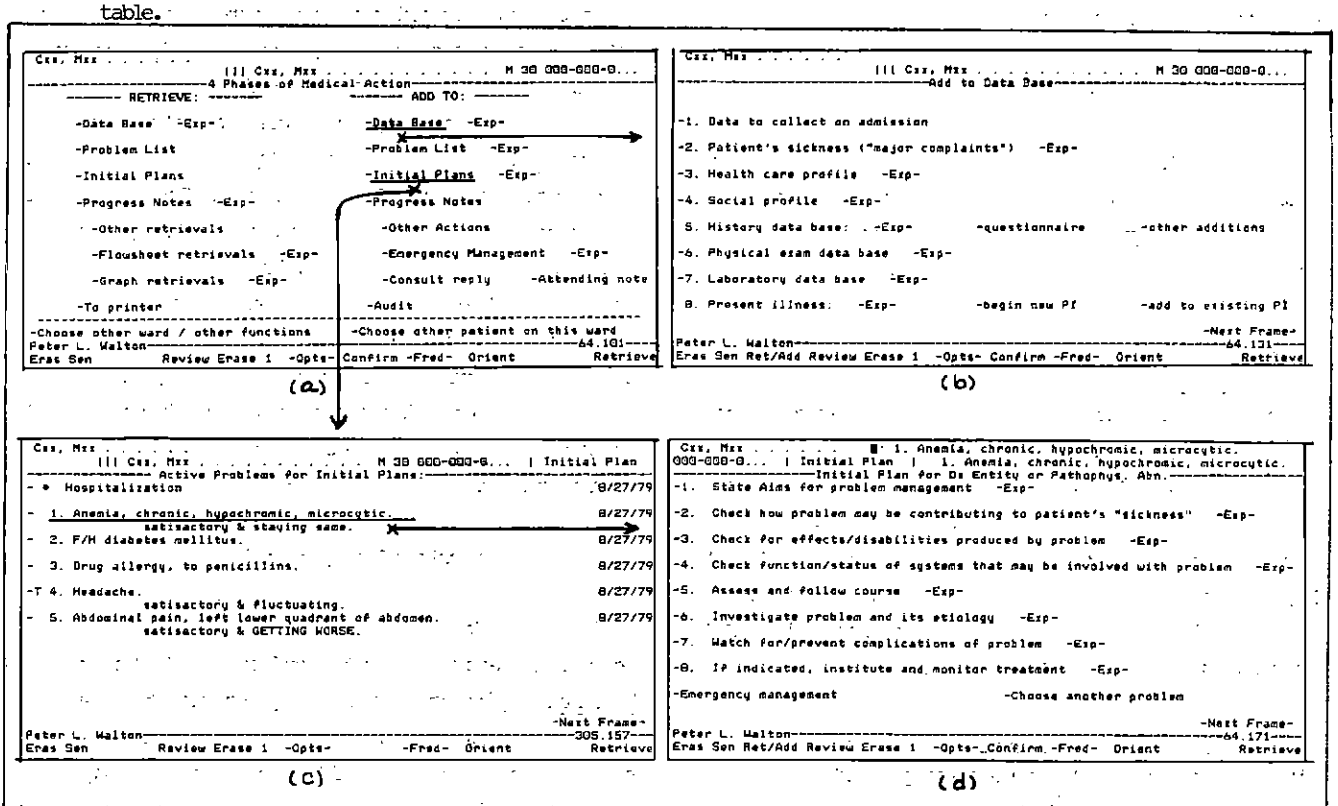


Figure 3. Example of guidance frames.

Frame (a) is the initial frame that was presented after a patient was selected from the list of patients whose records the user was allowed to access. From this frame, all additions to and retrievals from a single patient record can be performed. (Note the titles RETRIEVE and ADD TO near the top of the frame.) Selecting Data Base branches to the add to Data Base frame (b). Selecting Initial Plans, on the add side of (a), brings up a list of the patient's active problems. All patient care actions are taken in the context of a specific problem. If Anemia were selected, as in the example, the appropriate planning structure frame (d) is presented.

The displays contain frame content as well as context information for the user. The context changes as the user makes selections. The user's name is displayed in the lower left corner. Once a patient has been selected, the name is shown in the upper left hand corner. The selected problem is on the top line in the center.

12.3 Size and Maintenance of the Frame and Table Library

By 1979 more than 10,000 tables and 37,000 frames comprised the library. These frames and tables contained information on about 2700 medically defined problems, 640 drugs, 800 clinical laboratory procedures, 200 radiologic procedures, several hundred diets, activities, physical therapies and ward procedures, a comprehensive physical examination, patient history questionnaire, and problem-oriented superstructure.

Content was built in frames and tables from accepted texts and from articles in medical journals that satisfied selection criteria. More than 60 journals were regularly reviewed by medical librarians. Source, date, and builder of each piece of content was recorded in a reference. Auditing of existing content was performed continuously.

The frame and table library was maintained via an interactive frame and table editor (FRED – originally for FRame EDiting only), a syntax checker, a pathfinder, and other tools. The syntax checker reports violations of the rules used to build the frames and tables. The pathfinder finds all frames and tables that access a given frame or table. FRED manipulates the content in frames and tables. FRED is table driven: the type and form of each data element manipulated by FRED are specified in a set of tables accessed by FRED. FRED initiates backpointing and indexing among the frames and tables.

The frames and tables reference each other in order to define display branching or to use an entity. It is necessary to know how and where each paragraph is refereneed to effectively manage the frame and table library. All backpointing is performed by a routine scheduled by FRED.

12.4. Capturing Medical Knowledge

The medical content form and structure evolved from emphasis on providing guidance to capturing the information needed to build the guidance capability. From 1976 to 1979 Drs. Peter Walton, Robert Holland and Stuart Graves worked with the assistance of Larry Wolf, a software specialist, on means of capturing and codifying the medical knowledge. What evolved was the idea of capturing the basic units of medical knowledge, the facts (Walton, Holland, and Wolf 1979). A fact can simply state a relationship among entities or qualify a relationship in greater detail. As was stated:

> "By stating relationships, facts provide a means for structuring medical knowledge. The structure can be viewed as a network of relationships. In qualifying relationships, facts capture the content of medical knowledge.
>
> "Facts are the fundamental data base of medical knowledge and are structured in ways that facilitate access to the data base. For example, the fact in Figure 4 is a "cause fact." Cause facts state an etiologic relationship between a predecessor (iron deficiency) and a successor (transferrin saturation) when the predecessor is considered sufficient to explain the successor (i.e. iron deficiency "causes" a transferrin saturation of less than 16 percent). ... Predecessors and successors are used to define links which the system traverses to access the knowledge pertaining to a particular entity.

```
Fact 135.00303
     Ds: iron deficiency
     MF: transferrin saturation
     Value: below 16 % (average 7%)
     P[MFIDx] = 1

     Fact audited.
     Pr (coded): iron deficiency
     Su (coded): transferrin saturation

     Flags: p[Su|Pr] = 1;

     Refs:
       Clinical Hematology
       Wintrobe MM, et al., Eds.
       Philadelphia: Lea and Febiger, 7th ed., 1974
         Ch 16: Anemias characterized by deficient hemoglobin synthesis and
           impaired iron metabolism
         pp 621-634
     Process: 134.1  Dependence fact from authority source
     By: RR Holland.MD-SP   1/17/79
     Documentation audited.

     Refs:
       Clinical Hematology
       Wintrobe MM, et al., Eds.
       Philadelphia: Lea and Febiger, 7th ed., 1974
         Ch 17: Iron deficiency and iron-deficiency anemia
         pp 635-678
           Laboratory findings
           pp 656-660
     Process: 134.1  Dependence fact from authority source
     By: GW Gilroy RPh-SP   1/22/79
     Documentation not audited.
```

Figure 4. Fact in expanded text form.

"The ingredients needed to offer computer-based health-care guidance are now all in place. Knowledge is captured in atomic form in the facts. Facts are organized using a directed network. The network facilitates understanding and use of the facts to offer guidance. Building of this guidance capability is done in an explicit and rigorous fashion. Each step in the process of transforming knowledge to facts to guidance is documented and connected in electronic form to make both guidance and knowledge maintainable and adaptable."

13. The Master Library and Instances

The library system incorporated ideas outlined in the Conway report on the National Library of Displays. There were three classes of information in the generic library:

1. generic information that applied to all instances;
2. local information that applied to only one instance;
3. protocol information that may apply to several instances, but is designed and built with a specific set of values in mind.

Generic and protocol information were integrated into PROMIS by the library system. The generic library was responsible for maintaining all generic information and some protocol information. All other information was to be maintained locally by the instance that built and used it. All instance-specific data was identified in the PROMIS frame and table library, and procedures were implemented to allow the updating of only the generic information.

14. PROMIS Returns to a General Medical Ward

In December of 1976, the newly implemented PROMIS became operational on a 20 bed general medical ward (Brown III) in the Medical Center Hospital. The Brown III site was linked to the pharmacy, clinical laboratory and the radiology departments.

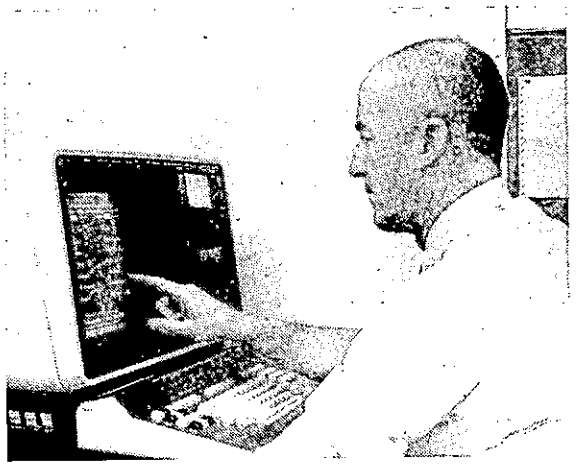
Functionally the new PROMIS was a major improvement over the CDC system. It was easier to use and to maintain. It had expanded functions and was more responsive and reliable. Until September of 1981 when PROMIS Laboratory left the University of Vermont and became PROMIS Information Systems, Inc., there were a steady stream of improvements and new functions. Below is a description of some of the features of the 1980 PROMIS system.

14.1. Addition of administrative functions

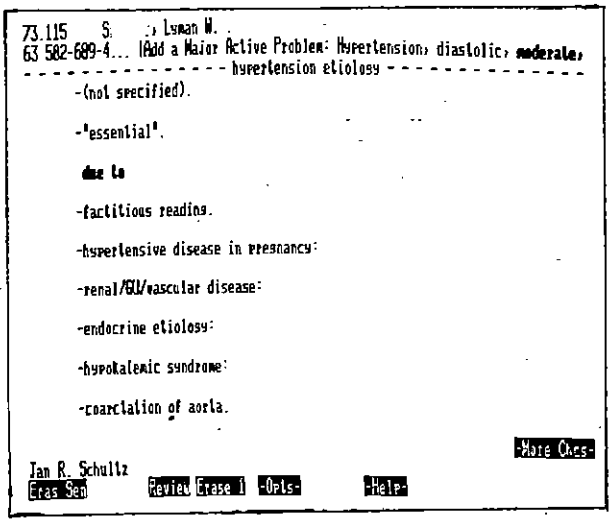
The administrative functions that are the basis for most computer based medical systems were not initially integrated into PROMIS. LeMay, along with other members of the PROMIS computer group, rewrote an existing administrative package to operate in conjunction with the electronic patient record. The resulting administrative system included patient billing, accounts receivable, accounts payable, and general ledger functions. All manual input was entered at a terminal. Admission, discharge, transfer, and charge data were generated automatically as a byproduct of use of the electronic record. Input to the general ledger from patient billing, accounts receivable, and accounts payable was generated automatically.

14.2 Networking was made routine

A two-node configuration was used routinely for PROMIS and system development; additionally, heavy simulated loads were applied to three nodes in order to validate the effects of contention for the basic network resources. Experiments were performed placing all terminals on one node and all medical records on another; this forced internode activity for each storage and retrieval operation. Under these conditions the user perceived little if any performance degradation due to the network.



Larry Weed using the Megadata Terminal.



Megadata Screen with inverse video function pads.

14.3. Population studies were added to PROMIS

The ability to interrogate the electronic patient records as a group for patient care or clinical research was provided by the population study system, developed by Stephen Reynolds. Population studies attempted to answer questions a clinical investigator might ask of the patient records. The population study system enabled three types of interrogation:

1)	Type of question:	Who has a specified combination of attributes?
	Form of result:	List of patients
	Example:	Who is receiving gentamicin?
2)	Type of question:	What are the results of a set of specified procedures?
	Form of result:	Array of values
	Example:	What is the standard deviation of BUN in patient group Y?
3)	Type of question:	What problems appear on the problem lists of patients in a specified group?
	Form of result:	Ordered list of problems
	Example:	What is the incidence of all problems identified in patients in group Z who have diabetes?

Peer Standards Review Organization personnel at the hospital performed audits using the population study tools, the results of which were submitted to the hospital administration and medical board. Questions were similar to:

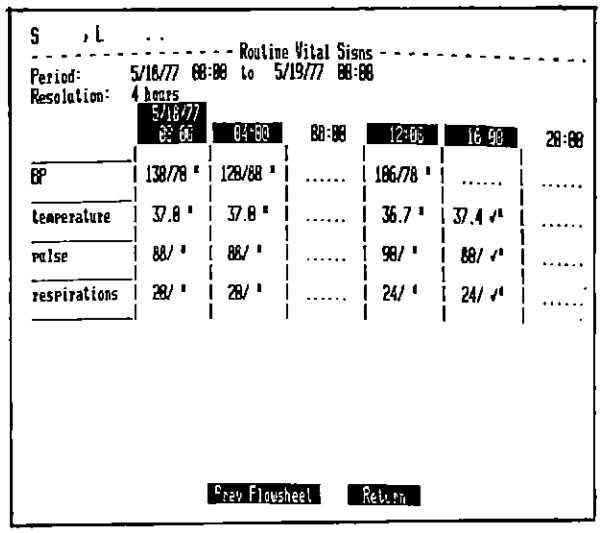
IF (date of serum creatinine) MINUS (date of digoxin) GREATER THAN 2 OR (date of digoxin) MINUS (date of serum creatinine) GREATER THAN 2
THEN "Serum creatinine not ordered in time"

The result of this query would be a list of patients for whom the assessment "Serum creatinine not ordered in time", as defined by the Boolean expression, was correct.

The processing and display of a patient set based upon the simple presence or absence of attributes was done while the user waited - usually within ten seconds or less; a patient set based on attributes qualified by time or other variables took longer depending on the complexity of the question and the number of patient records processed.

It was also possible to pass data from the electronic patient records to the MINITAB II statistical package (developed by the National Bureau of Standards and modified at the University of Pennsylvania) for statistical analysis. MINITAB was implemented with the commands as choices on frames, allowing easy, rapid, controlled and structured access.

All patient attributes within the electronic records that were used for population studies had to be coded. Complex relationships within one procedure report could not be captured. A coding mechanism was needed to parallel the text string seen by the user. The development of the code string, a tree-structured, frame-driven encoding mechanism, would have to wait for the next application design iteration.



Example of vital signs flowsheet for a single day.

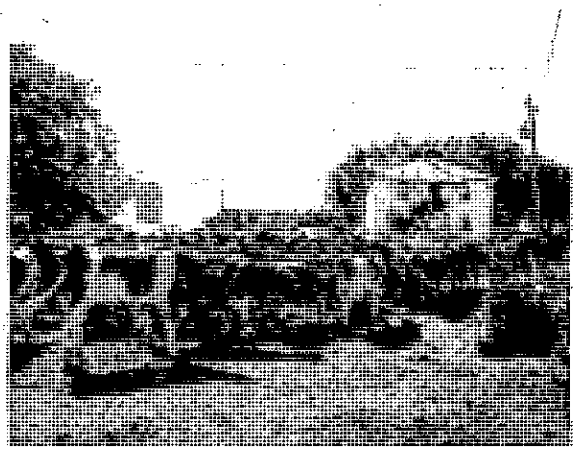
14.4. Patient Lifetime Records were developed

The structure of the electronic record was expanded to allow multiple inpatient admissions and outpatient visits within one record. The encounter was the structuring mechanism for the patient's lifetime record. Each inpatient admission or outpatient visit was a new encounter. The user could specify any encounter (or all encounters) from which to retrieve patient data.

14.5. Communications System Use

The MEGADATA touch-screen terminals, as well as printers and other peripherals, were connected to the cpu with a high-speed communications bus. The data communications lines could be extended up to 20 miles (32 km) by cable and further extended with microwave links. A very high frequency (VHF) 2-way coaxial bus was employed to provide many

inexpensive interconnect ports. Full-duplex modems operated at 307,200 bits per second. Programmable controllers managed peripheral devices and operated at the remote interconnect ports. The line control protocol was a simple polling protocol. Each remote device was assigned a unique poll address. Because of the line protocol and the nature of the bus, users were free to move terminals and other peripherals from one location to another and continue without any access (sign-on) requirements. (Wanner 1978). The bus was extended with a microwave link to the Grand Isle Medical Clinic 17 miles from the Hospital.



The Grand Isle Clinic. Find the microwave dish?

15. Conclusion

15.1 The Human Interface

The combination of a response rate of less than .25 second and the large data base of frames allows users to navigate through the network of frames very rapidly, accomplishing their tasks without using the keyboard. The interaction rate is very high since a single selection results in the presentation of a full screen of new text. A complete thought can be communicated in a single touch. "The data base available to a user is a combination of electronic records generated from past selections and of frame and table knowledge. The user can operate from a universe larger than what can be remembered and has only to recognize the correct selection (and not recall it). These features provide for a unique and very effective human-computer interface.

The building of the data base of frame and table knowledge is an intellectually deep task and is time consuming. The PROMIS frame and table library took over 200 person-years of effort to build. This type of interface can be successful if the cost of developing and maintaining the knowledge base used by the interface can be spread across many users.

In order for the interface to work, it requires an extensive knowledge base as well as a different relationship to the computing resources it uses. For PROMIS, the interface is the computation and not a means to get to a computation. As computing power gets less expensive, this interface becomes more cost effective.

Robertson, Newell, and Ramakrishna discuss the PROMIS/ZOG type of interface and claim that this "type of interface is a preferred mode of man-machine interaction, even over the use of natural language dialogue with the computer in the role of intelligent agent.

"There are two polar views about how to structure man-machine interaction. One is the computer as tool. ... Control remains with the user. The other view is the computer as intelligent assistant. In this view one wishes to make the computer more intelligent and communication with it more natural ... Precisely what an intelligent assistant is supposed to provide is freedom (of the user) from the effort of understanding. Put one other way, delegation requires an act of faith.

"ZOG is an evolution in the tool direction. It seeks to produce a transparent device which, in itself, has no intelligence at all, but is immensely responsive to the user. It seeks to do this in the arena where we normally expect to use natural language, namely, dealing with large bodies of knowledge. Indeed ZOG uses natural language for its output (though arranged in a sort of spatial dialogue), for the user has good devices for assimilating it. However, its own internal structure, which governs what it says and when it chooses to say it, is completely open to examination by the user. (Robertson 1977)

15.2. Application Development

In October of 1981, PROMIS Laboratory left the University of Vermont and formed PROMIS Information Systems, Inc. We realized that the potential for the PROMIS technology was broader than medicine alone, and that work was needed on the application development software. From the very beginning of our development in 1967, the Human Interface

Program interpreted the frame in the process of bringing up the next display. It was table driven since the frame (the programs table) defined how the program was to operate. Other parts of the system were not table driven. The frame editor has been table driven since 1978, but the storage and retrieval programs had imbedded within them the logic of the Problem Oriented Medical Record and could handle only one type of record structure. We have rewritten the application subsystems so that they are all table driven. This allows the development of systems as large and complex as the PROMIS medical application without writing PPL programs.

Application development without writing programs makes programmers more efficient by a factor of between 10 and 100. What we have developed is not a programmer-less system, but a set of tools that makes programmers more effective. The tool kit uses the concepts of programming to accomplish the tasks of programming, but the writing of procedural programs is not required.

15.3. The Future

When we began our work, a touch-screen terminal operating over telephone lines at 2400 bits per second cost over $20,000. Today, a similar terminal could be purchased off-the-shelf for under $3,000. The personal computer I'm using to write this paper has more central memory in it then the CDC 1700 system used for the initial development and the personal computer's central memory cost 200 times less. The technology is available now to make the type of interface and systems described in this paper cost effective and widely available.

The PROMIS system is alive and kicking. It is still the most advanced medical information system in existence and the only one to manage a fully electronic medical record. It has yet to be used as the information system for a total hospital, but its time is coming.

Acknowledgments

This paper represents the efforts of many individuals in medicine and in computer science, both within and outside the past PROMIS Laboratory. I would like to acknowledge the group from PROMIS Laboratory who have stayed together to keep these ideas alive. This work was funded primarily by the National Center for Health Services Research, Department of Health, Education, and Welfare.

References

(Hertzberg, Schultz, Wanner 1980) Hertzberg, R.Y., Schultz, J.R., Wanner, J.F.
The PROMIS Network
Computer Networks 4, 1980 pp. 215 - 228

(Parnas 1972) Parnas, D.L.
A Technique for Software Module Specification with Examples
Comm. ACM 15(5), pp.330-336 (May 1972)

PROMIS Laboratory Staff
The Representation of Medical Knowledge
Proc. Second Annual Symposium on Computer Applications in Medical Care, Washington, DC, Nov. 1978, pp 368-400.

(Robertson 1977) Robertson, G., Newell, A., Ramakrishna, K.
ZOG: A Man-Machine Communication Philosophy
Department of Computer Science, Carnegie-Mellon University, August 4, 1977.

(Schultz, Davis 79) Schultz, J.R., Davis, L.
The Technology of PROMIS
Proc. of IEEE, Vol. 67, 9, Sept. 1979.

(Schultz, Cantrill, Morgan 1971) Schultz, J.R., Cantrill, S.V., Morgan, K.G.
An Initial operational problem oriented medical record system - for storage, manipulation and retrieval of medical data
AFIPS - Conference Proceedings, Vol 38, 1971.

(Walton, Holland, Wolf 1979) Walton, P.L., Holland, R.R., Wolf, L.L.
Medical Guidance and PROMIS
IEEE, Computer, November, 1979, pp 19 - 27.

(Wanner 78) Wanner, J.F.
Wideband Communication System Improves Response Time
Computer Design, Dec. 1978, pp. 85 - 91.

(Weed 64) Weed, Lawrence L., MD.
Medical Records, Patient Care and Medical Education
Ir. J. Med. Sc., June 1964, pp. 271 - 282.

(Weed 68) Weed, Lawrence L., MD.

Medical Records that Guide and Teach
New England Journal of Medicine, 278:593-600, 652-657
(March 14, 21), 1968.

(Weed 69) Weed, Lawrence L., MD.
Medical Records, Medical Education and Patient Care
Case Western Reserve Press, Cleveland, Ohio, 1969.

User Technology: From Pointing to Pondering

Stuart K. Card and Thomas P. Moran
Xerox Palo Alto Research Center

From its beginning, the technology of personal workstations has been driven by visions of a future in which people would work in intimate partnership with computer systems on significant intellectual tasks. These visions have been expressed in various forms: Memex (Bush, 1945), Man-Machine Symbiosis (Licklider, 1960), NLS (Engelbart, 1963), Dynabook (Kay, 1977), and others.

The tight coupling between human and computer required by these visions necessitated advances in the ways humans and computers interact. These advances have slowly begun to accumulate into what might be called a *user technology*. This user technology includes hardware and software techniques for building effective user interfaces: bitmapped displays, menus, pointing devices, "modeless" command languages, animation, and interface metaphors. But it must include a technical understanding of the user himself and of the nature of human-computer interaction. This latter part, the scientific base of user technology, is necessary in order to understand why interaction techniques are (or are not) successful, to help us invent new techniques, and to pave the way for machines that aid humans in performing significant intellectual tasks.

In this paper, we trace some of the history of our understanding of users and their interaction with workstations—the personal part of personal workstations. In keeping with the spirit of other papers at this conference, we have centered this review around our own experiences, perspectives, and work and have not attempted a complete history of the field. In concentrating on our own work, we do not wish to mimimize the importance of others' work; we simply want to tell our own story. Our focus is on what we have learned about users in our years of studying them and how we see our findings relating to the original visions of the personal workstation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1986 ACM-0-89791-176-8-1/86-0183 $00.75

1. The Vision of an Applied User Psychology

The opportunity to tackle a science of the user brought us to PARC in 1974 (collaborating with Allen Newell, as consultant). As other PARC researchers were beginning to pursue the vision of highly graphic, interactive, network-based personal workstations, we were following a vision of our own. The idea was to draw concepts from cognitive psychology and artificial intelligence to create an applied cognitive science of the user. We called our project the Applied Information-processing Psychology Project (AIP). A 1971 memo by Allen Newell proposing this project to PARC stated the basic argument this way:

> (1) There is emerging a psychology of cognitive behavior that will permit calculation of behavior in new situations and with new humans. . . .
>
> (2) Several of the tasks that are central to the activities of computing—programming, debugging, etc.—are tasks that appear to be within the early scope of this emerging theory.
>
> (3) Computer science in general is extremely one-sided (for understandable reasons) in the treatment of its phenomena: almost no effort goes into understanding the nature of the human user. . . .
>
> (4) There is a substantial payoff (in dollars) to be had by really designing systems with detailed understanding of the way the human must process the information attendant thereto.

In 1974, we were in the position of having to create a new field. Psychological theories and methodologies held the promise of being able to represent and manage complex cognitive tasks, but the only body of research pertaining to human-computer interaction was in the field of human factors, where studies were largely empirical and evaluative, concentrating mostly on sensory-motor questions like the best shape for a switch. Human-computer interaction, involving two active agents each capable of initiating an exchange of information, inherently involves human

cognitive processing. Our vision was to create a science of the user rooted in cognitive theory. But we also wanted the science to be practical, providing the system designer with the conceptual tools to think about the key characteristics of the user and the calculational tools to take account of the user's behavior.

To simplify a rather complex history for the purposes of this paper, we will narrate just a few strands of what unfolded. Each strand is organized around a particular aspect of the interface between the user, the system, and the task. The strands are organized into a sequence of ever-broader interfaces, from the physical interaction of the user with devices and displays to the symbiotic interaction of the user with the system while grappling with complex intellectual tasks—that is, from pointing to pondering:

> *The physical interface:* The user interacts with a system by means of physical input devices, such as a keyboard and a mouse, and output devices, such as a high-quality graphical display.
>
> *The cognitive interface:* The user has certain characteristics as an information processor, such as a limited working memory, that together with the goals he is trying to achieve determine his behavior.
>
> *The conceptual interface:* The computer system is also a complex information processing mechanism, and the the user needs to have some kind of mental model of it in order to effectively interact with it.
>
> *The task interface:* Systems are designed to help their users do tasks, not only small, routine tasks, but also larger, difficult intellectual tasks that are the object of the grand visions of personal workstations.

The story within each strand of our work is chronological; but the first three strands, representing our previous work, overlap in time. The last strand represents our current work.

2. The Physical Interface: Pointing

Bill English, who had among other thing engineered and built the mouse for Doug Engelbart (English, Engelbart, and Berman, 1967), was one of the people who migrated to PARC when it was founded. We worked for English when we first came to PARC. Engelbart and English had always considered the mouse to be an interim device, and English wanted to see if it was possible to invent other devices that would improve on its speed. In particular, he was interested in whether one could put a key on the keyboard that would sense force and direction without requiring the user to remove his hands from the keyboard. He had a number of devices built, and we helped design an experiment to test them. The results (Figure 1) showed that none of the devices tested improved on the mouse either for speed or for error rate (Card, English, & Burr, 1978).

This was a typical human factors experiment: we compared a set of systems to determine which one performed best. But this direct empirical comparison between devices was just the sort of methodology of human factors testing that we wanted to improve: we wanted to understand the reasons why the results came out the way they did. We therefore made mathematical models of each device and tested them against the data until we found models that fit. The model for the mouse was particularly instructive. The mouse was best modeled by a version of Fitts's Law:

$$Movement\ Time = Constant + .1 \log_2(D/S + .05)\ \text{sec},$$

		Movement Time for Non-Error Trials						Error Rate	
		Homing Time		Positioning Time		Total Time			
Device	Trials N	M (sec)	SD (sec)	M (sec)	SD (sec)	M (sec)	SD (sec)	M	SD
Mouse	1973	.36	.13	1.29	.42	1.66	.48	5%	22%
Joystick	1869	.26	.11	1.57	.54	1.83	.57	11%	31%
Step Keys	1813	.21	.30	2.31	1.52	2.51	1.64	13%	33%
Text Keys	1877	.32	.61	1.95	1.30	2.26	1.70	9%	28%

Figure 1. Overall pointing times for all devices. Homing Time is the time to move the hand from the keyboard to the device, Positioning Time is the time to move the cursor to the target. Times are based on a standardized set of distances and target widths. Averages are computed on the basis of four users x 600 trials/users. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.4) with the permission of Lawrence Erlbaum Associates, Inc.*

where D is the distance the hand moves to the target and S is the target width (see Figure 2). The significance of this result is that this is the same law that describes movement time for the hand alone, with about the same constant of proportionality. The limiting factor in moving the mouse, therefore, is not in the mouse, but in the eye-hand coordinate system itself. That, in turn, means the mouse is nearly optimal, at least with respect to the set of muscles used. Therefore, designing a device that is faster than the mouse would be difficult.

Here was a prototypical example of the kind of theory we wanted to build—a model precise enough to enable designers to perform back-of-the-envelope calculations, a model that identified key constraints in the design space for pointing devices. With this model we could be sure that the data giving comparisons among devices would probably generalize to new situations, because we knew the main factors that governed the results. Furthermore, the model provided guidance for interface designers: make distant buttons large, for example. These studies were heavily used in the debate within Xerox that led to the decision to depart from tradition by including a mouse with the new Star product.

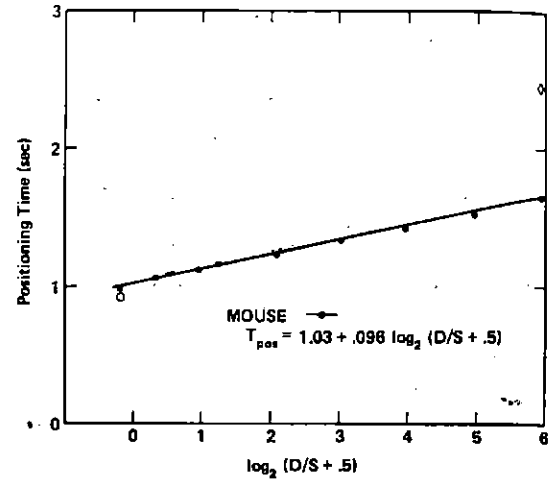


Figure 2. Positioning time for the mouse as a function of Fitts's Index of Difficulty. The fit of positioning time data to the straight line in the figure shows that time for the mouse is governed by Fitts's Law (is proportional to Fitts's Index of Difficulty). The constant of proportionality, .096 sec/bit (= 10.4 bits/sec), is approximately the same value as for pointing with the hand alone. *Reprinted from Card, Moran, & Newell (1983, Fig. 7.8) with the permission of Lawrence Erlbaum Associates, Inc.*

3. The Cognitive Interface: Cognitive Skill

The Model Human Processor

An interesting result of the mouse study was the way an evaluation of pointing devices led to a consideration of human information processing characteristics. This experience pointed to the need for an engineering model of users that would summarize such characteristics. We knew of many phenomena scattered in the literature of psychology, such as Fitts's law, that would be helpful for system design.

To someone who is not a specialist, such as a designer, this literature appears disorganized and contradictory. Psychologists love to split hairs and find small contradictions in published models. The robust but approximate generalizations that might be made to work for engineering tend to get trampled in the debates. Although we had the notion of such a model from about 1974, it wasn't until 1982, when we were nearing the completion of our book, that we were able to formulate it. The model, called the *Model Human Processor*, was inspired by the processors, memories, and switches (PMS) notation of Bell and Newell (1971) for describing the architecture of computing systems. It was a simplified architecture of the user, described in terms of three processors, four memories, 19 parameters of these, and 10 principles of operation (Figure 3).

As an example of the sort of calculation that can be done with the Model Human Processor, consider the case where a programmer is programming a video game version of billiards. He needs to know how long he has after the collision of two balls to compute the balls new trajectory before the illusion of causality breaks down. The Model Human Processor tells us that every event that occurs within 100 msec will be perceived as a single event, so a rough estimate is 100 msec. But it also recognizes that second-order phenomena can change this number, so it also supplies a range of uncertainty for this number (in this instance, 50 to 200 msec).

In this case, we can say that if the programmer can make the balls move within the 50 msec lower bound for this parameter, then it is pretty certain that users will perceive the collision as the cause of the balls change in direction, regardless of secondary effects such as brightness or contrast ratio of the screen.

The Model Human Processor can be used to compute predictions about human performance: how fast people can read, how fast they can scribble, the effect of different abbreviation schemes on memory error, and so on. Derivations based on the Model Human Processor were used to set the maximum velocity of the mouse on the workstation for the Xerox Star.

Text Editing and Cognitive Skill

The Model Human Processor also contains, as one of its principles of operation, Herbert Simon's bounded rationality principle. Our version of this principle may be stated:

> *A person acts so as to attain his goals through rational action, given the structure of the task, his*

inputs of information, and bounded by limitations upon his knowledge and processing ability. (Card, Moran, & Newell, 1983).

That is, in order to predict a person's behavior, one needs to analyze the task he is trying to do, because the person will simply do what is rational to accomplish the task, constrained by limitations in his knowledge and ability to process information. Studies of how people solve problems had shown (Newell & Simon, 1972) that their behavior could be modeled as a search through a space of states of the problem, a *problem space.* We expected that in studies of human-computer interaction we would find users searching through problem spaces to accomplish goals, trying various solutions, backing off and taking other tacks when they ran into trouble.

Two early decisions we made led us to results which were contrary to our expectations. Although we had come to PARC initially with the intention of studying computer programming, we decided once we arrived that there were strategic advantages to studying text editing. A second, tactical, decision was to work with expert subjects rather than novices, in order to have more stable behavior to analyze. Further, to make sure that all our subjects had the

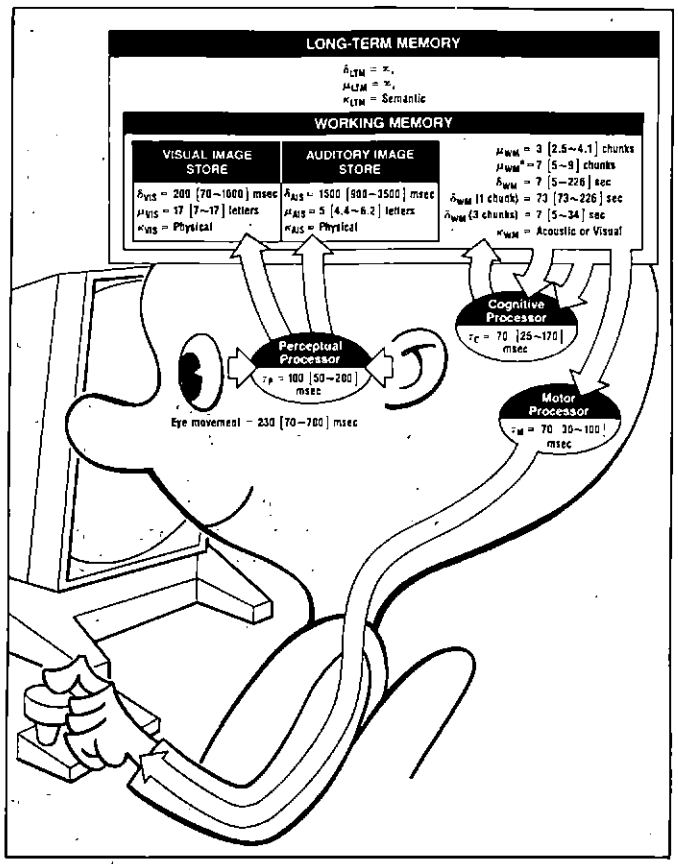


Figure 3. The Model Human Processor memories and processors. Depicted schematically in the figure are the memories, processors, and constants used for making simple computations. The basic architecture of the model can be summarized thus: Sensory information flows into Working Memory through the Perceptual Processor. Working Memory consists of activated chunks in Long-Term Memory. The basic principle of operation of the Model Human Processor is the Recognize-Act Cycle of the Cognitive Processor: On each cycle of the Cognitive Processor, the contents of Working Memory initiate actions associatively linked to them in Long-Term Memory; these actions in turn modify the contents of Working Memory. The Motor Processor is set in motion through activiation of chunks in Working Memory.

Predictions are made using time constants from the figure and a set of associated Principles of Operation: (P0) The Recognize-Act cycle of the Cognitive Processor; (P1) The Variable Perceptual Processor Rate Principle; (P2) The Encoding Specificity Principle; (P3) The Discrimination-Principle; (P4) The Variable Cognitive Processor Rate Principle; (P5) Fitts's Law; (P6) The Power Law of Practice; (P7) The Psychological Uncertainty Principle; (P8) The Rationality Principle; and (P9) The Problem Space Principle. *Reprinted from Card, Moran, & Newell (1983, Fig. 2.1.) with the permission of Lawrence Erlbaum Associates, Inc.*

same goals in mind, we presented them with a fairly explicit task—the "manuscript editing task"—which required them to work from a marked-up manuscript, making the changes explicitly indicated.

When we analyzed videotaped protocols of our subjects doing this task, we were surprised to find little of the search behavior of problem solving that we had expected. Subjects simply looked at the tasks and did them. The tasks were not problematic. The subjects had done similar tasks many times before, and had built up a large repertoire of methods that could be applied to the tasks. This wasn't problem solving; we came to call it *cognitive skill* and set out to build models to characterize and predict this mode of behavior.

We applied our theories of human information processing to the kind of specific skills necessary for the text-editing task. The result was a class of models in which the user's cognitive structure consists of four components: (1) a set of familiar Goals that the user would recognize when faced with a specific task; (2) a set of primitive Operators (actions) that the user was skilled at performing and could deploy whenever necessary; (3) a set of Methods, consisting of "compiled" sequences of subgoals and operators, that the user could use to attain his goals; and (4) a set of Selection rules that enable the user to choose among competing methods for goals. We call a model specified by these components a *GOMS model* (Figure 4 shows an example of one). Together, these components constitute the user's cognitive skills for performing tasks. If a user has enough knowledge of this kind, it isn't necessary to use problem-solving strategies. All that is needed is to examine the task, characterize it in terms of a specific goal, select the appropriate method, and then execute it.

In order to test our GOMS models, we ran a set of experiments to determine whether we could explicitly specify this kind of GOMS knowledge and thus predict what users would actually do. We also wanted to learn how the degree of resolution—the grain of specified detail of such a model—affects the degree of accuracy of predictions based on the model. We expected fine-grained models to yield increased accuracy. Knowing that it takes a lot more work to develop a fine-grained model, a practical question related to the applicability of our models was whether the additional work of constructing a fine-grained model was worth the effort.

To find out, we built a family of models in the GOMS framework that characterized the behavior of users of a specific computer text-editing system and ran experiments to test our predictions from these models. We were surprised to discover that fine-grained models did not yield a worthwhile or even a significant increase in prediction accuracy (Card, Moran, & Newell, 1976, 1980a). More felicitous, practically speaking, was our discovery that even the crude models seemed to capture and predict behavior fairly well. These properties suggested that the GOMS model could be turned into the kind of engineering tool that a designer could use to model and predict skilled user behavior in computer-mediated tasks. That was what we proceeded to try next.

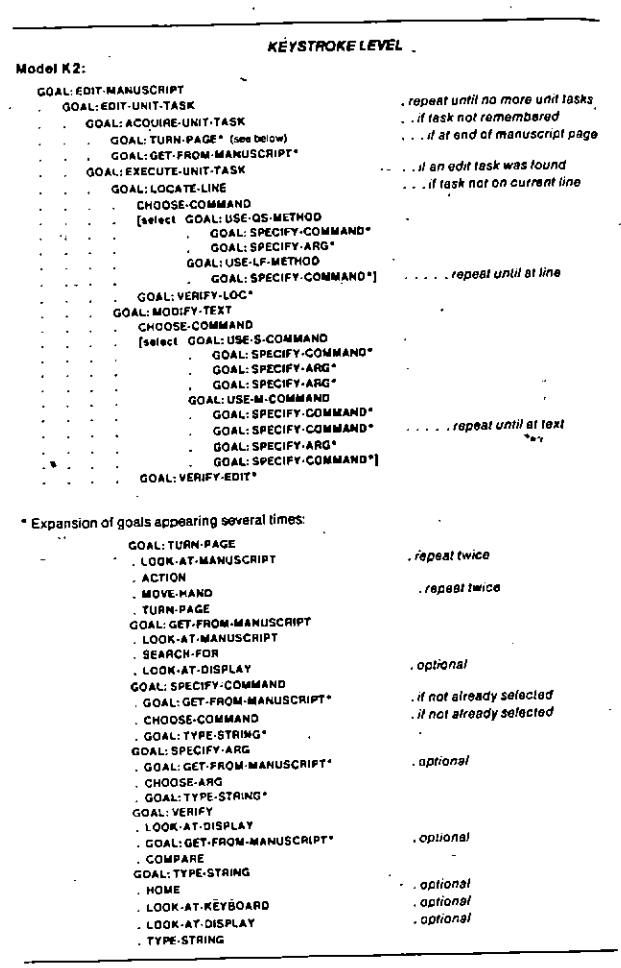


Figure 4. GOMS Model K2 for the text editor POET. This model has a grain of about 0.5 sec/operator. The user is using a line-oriented editor to make changes to a computer file previously marked on a paper manuscript. *Reprinted from Card, Moran, & Newell (1983, Fig. 5.12) with the permission of Lawrence Erlbaum Associates, publishers.*

The Keystroke-Level Model

In simplifying the GOMS models into an engineering model that we could hand to a designer, we constructed an idealized prediction problem:

> *Given a task (possibly involving several subtasks), the command language of a system, the motor skill parameters of the user, the response time parameters of the system, and the method used for the task, predict how long an expert user will take to execute the task using the system, providing he uses the method without error.* (Card, Moran, & Newell, 1980b)

The Keystroke-Level Model enables a system designer to make such predictions with a "back-of-the-envelope" style of calculation.

To do the calculation, the designer codes the method a user employs to do a task in terms of a set of operations derived from one of the fine grain GOMS models (Figure 5). In this simplified model, all keystrokes are assumed to take a constant amount of time; and pointing with the mouse is also assumed to take a constant amount of time. Mental activity by the user is reduced to a single generic mental preparation operation, and rules governing when it will occur are provided.

A typical use of the model to analyze a method is given in Figure 6. This method is one way in which a hypothetical text-editor could be used to replace a word recently typed by the user. Each action of the method is described in terms of the operators of the model, then a time for the method is computed. In this case, the method is expected to take the same amount of time regardless of how many words back the word to be replaced is located. Figure 7 shows the expected time for performing this methods and two other methods available in the editor as a function of the number of words back the to-be-changed word is located. It can be seen that the time profiles of the three methods are quite different and that each is fastest at different times.

Operator	Description and Remarks	Time (sec)
K	PRESS KEY OR BUTTON. Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	.08
	Good typist (90 wpm)	.12
	Average skilled typist (55 wpm)	.20
	Average non-secretary typist (40 wpm)	.28
	Typing random letters	.50
	Typing complex codes	.75
	Worst typist (unfamiliar with keyboard)	1.20
P	POINT WITH MOUSE TO TARGET ON A DISPLAY. The time to point varies with distance and target size according to Fitts's Law, ranging from .8 to 1.5 sec, with 1.1 being an average. This operator does *not* include the (.2 sec) button press that often follows. Mouse pointing time is also a good estimate for other efficient analogue pointing devices, such as joysticks (see Chapter 7).	1.10
H	HOME HAND(S) ON KEYBOARD OR OTHER DEVICE.	.40
D(n_D,l_D)	DRAW n_D STRAIGHT-LINE SEGMENTS OF TOTAL LENGTH l_D CM. This is a very restricted operator; it assumes that drawing is done with the mouse on a system that constrains all lines to fall on a square .56 cm grid. Users vary in their drawing skill; the time given is an average value.	$.9n_D + .16l_D$
M	MENTALLY PREPARE.	1.35
R(t)	RESPONSE BY SYSTEM. Different commands require different response times. The response time is counted only if it causes the user to wait.	t

Figure 5. The operators of the Keystroke-Level Model. The figure lists the operators needed to analyze user interface methods and to make calculations of user performance with these methods. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.1) with the permission of Lawrence Erlbaum Associates, Inc.*

Even though our GOMS study suggested that simple models could be effective, and even though the Keystroke-Level Model was a careful simplification of one of the GOMS models, we felt that it was necessary to rigorously text the explicit performance assumptions of the model. To validate the Keystroke-Level Model, we ran a large set of experiments in which people performed tasks with text editors, graphics editors, and executive

Method R (Replace):

Terminate type-in mode	**MK**[ESC]
Point to target word and select it	**H**[mouse] **P**[word] **K**[YELLOW]
Call Replace command	**H**[keyboard] **MK**[R]
Type new word	**4.5K**[word]
Terminate Replace command	**MK**[ESC]
Point to last input word and select it	**H**[mouse] **P**[word] **K**[YELLOW]
Re-enter type-in mode	**H**[keyboard] **MK**[I]

$$T_{execute} = 4t_M + 10.5t_K + 4t_H + 2t_P$$
$$= 12.1 \text{ sec}.$$

Figure 6. Encoding of Method R. Use of the Keystroke-Level Model to describe one possible method in a mouse-based display editor for replacing a word previously mistyped. It is assumed the word is still visible on the screen. *Reprinted from Card, Moran, & Newell (1983, p. 289) with the permission of Lawrence Erlbaum Associates, Inc.*

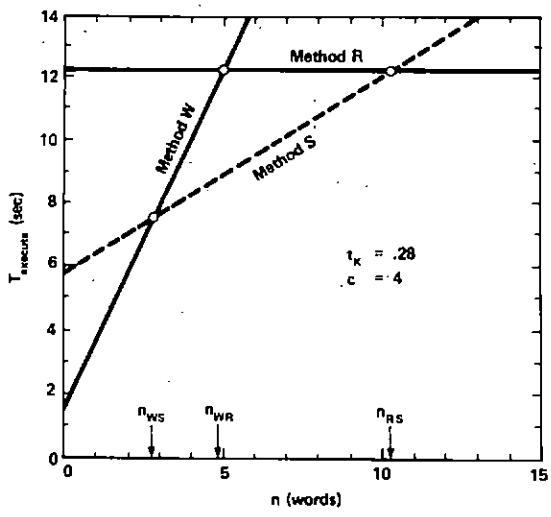


Figure 7. Execution time of three methods for the misspelled-word task as a function of *n*. This figure shows the use of the Keystroke-Level Model for the parametric comparison of three different methods for accomplishing the same goal. In this case each method appears to be superior in a certain range of *n*. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.12a) with the permission of Lawrence Erlbaum Associates, Inc.*

subsystems. The results (Figure 8) showed a good fit between predictions derived from the model and observed times required for the tasks. Therefore, our simple approximative model would be of some use in making practical time estimates.

The Keystroke-Level Model has actually proved useful in real system design. One application was the determination of the number of buttons on the mouse for the Xerox Star product. Several schemes for selecting text in the Star text editor were proposed. These schemes required different numbers of buttons. The goal was to make a mouse with the smallest number of buttons possible, so that it was easy to learn to operate. Experiments to test the schemes were reasonably easy to run with novice subjects. Everyone is a novice subject on a new system, and being a novice doesn't require training. But understanding how well the schemes would work for expert users (which most users would be for most of their time on the system) was expensive, because a long time would have to be spent training the users. The solution was to run experiments for novices and to use the Keystroke-Level Model for predicting expert performance.

The Keystroke-Level Model allowed us to carry through, at least in regard to a very specific kind of behavior, part of our original vision of packaging psychological knowledge into a model that designers can use to calculate user performance with a variety of interactive computer systems.

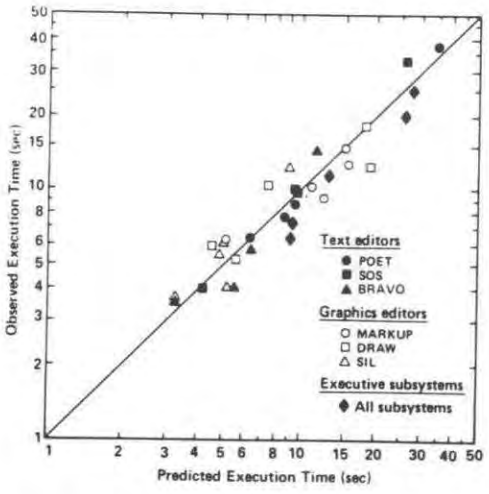


Figure 8. Predicted vs. observed execution times in tests of the Keystroke-Level Model. Predicted execution times were calculated from the Keystroke-Level Model. Observed execution times come from empirical observation. *Reprinted from Card, Moran, & Newell (1983, Fig. 8.6) with the permission of Lawrence Erlbaum Associates, Inc.*

The Unit Task

Not all skill characteristics can be reduced to simple counting, however. In studying text editing, we observed a characteristic of cognitive skill that is fundamentally cognitive, namely the organization of user behavior into short, quasi-independent tasks, which we call *unit-tasks.* Figure 9 shows a typical timeline of user behavior, in this case from a protocol of an electrical engineer using a graphic CAD system to design a VLSI circuit. The figure shows that the behavior is divided into chunks, each consisting of a a few seconds of pause (to formulate a task to do) followed by a few seconds of activity (to execute the task). Each think-execute chunk of behavior is a unit task.

The unit task structure of cognitive skill is interesting because the performance limitations of the user show through the purely rational organization of his behavior. The most significant reason why the unit task breakdown of behavior arises is because of limitations in the user's working memory. If the user can manage input and output streams in his working memory, then the user's behavior will have a continuous structure, as in touch typing from a manuscript. But, when conditions on the inputs and outputs do not allow this, then behavior must be structured into a series of unit tasks.

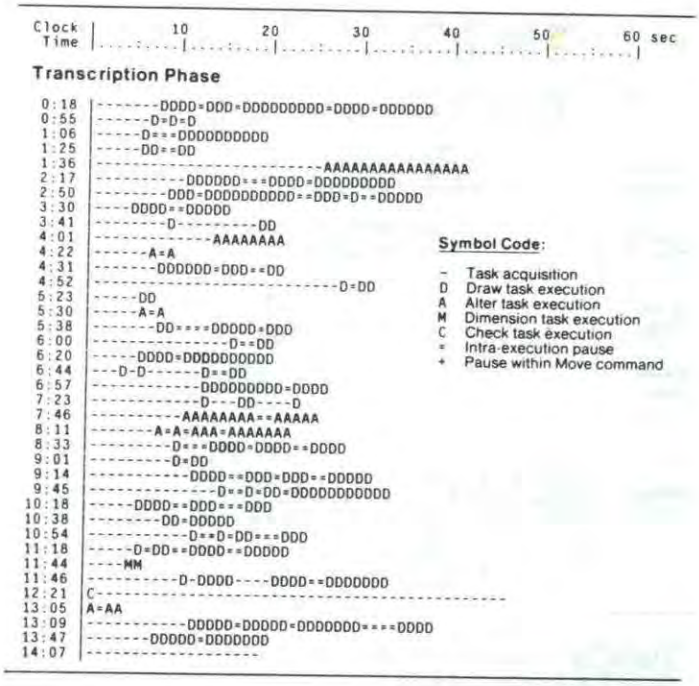


Figure 9. Time line representing the user's behavior sequence in a VLSI design session. Each single-character symbol represents one second of behavior. The symbol sequence begins on a new line at the beginning of each unit task, and the clock time is the time at the beginning of the unit task. *Reprinted from Card, Moran, & Newell (1983, Fig. 10.7) with the permission of Lawrence Erlbaum Associates, Inc.*

Figure 10 portrays a trace of the working memory load of the user, based on logical considerations of when information must be available in memory in order to do a task. As we can see, memory requirements build from a low point at the start of a unit task to a high within the unit task, where information for the task has been assembled, and back to a low point at the end of the unit task, where the information is no longer needed. If the information in working memory should reach a level higher than the working memory capacity, then user performance will suffer, usually manifested by the user committing errors. To avoid these errors, the user must break down the overall task into smaller (unit) units, each of which can be managed in the available working memory.

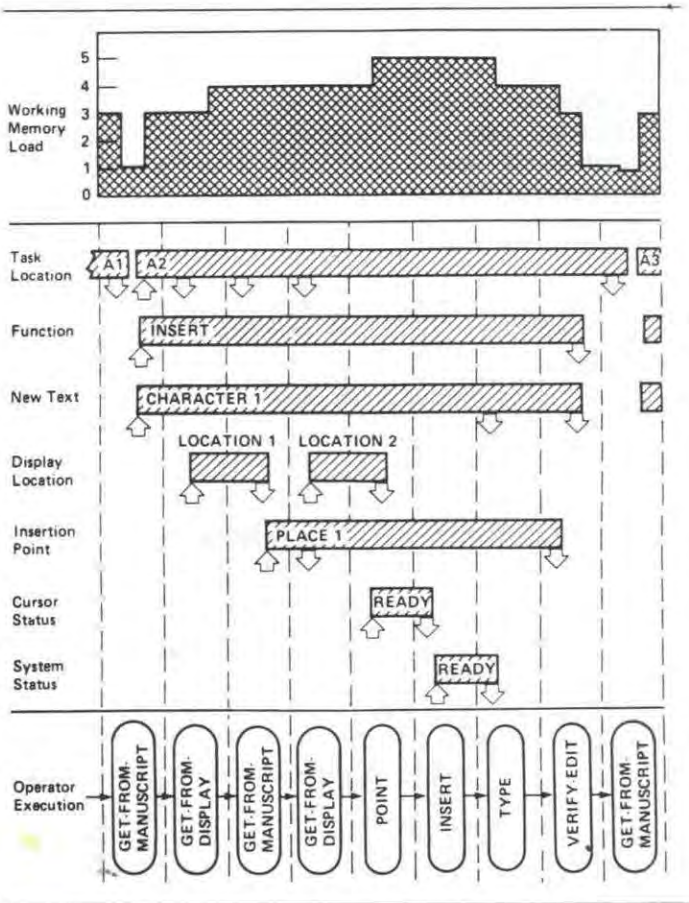


Figure 10. Data in Working Memory during a unit task. This figure is an hypothetical trace of the performance of one unit task. Time runs to the right on the horizontal axis. The bars indicate the time during which each piece of information about the task is needed. The arrows indicate the initial time data is available and the subsequent times it is needed. The histogram on the top plots the total Working Memory load over time, showing how the load peaks within a unit task and dips between unit tasks. *Reprinted from Card, Moran, & Newell (1983, Fig. 11.12) with the permission of Lawrence Erlbaum Associates, Inc.*

Unit-task behavior is at present a theoretical notion based on empirical observation (see Card, Moran, Newell, 1983, for further discussion). But it represents an important feature of user behavior—a sort of "cognitive rhythm"—that should be taken into account in designing user-computer interfaces that are sensitively tuned to the user's capabilities.

The Psychology of Human-Computer Interaction

In 1978, one of us had completed a doctoral thesis (Card, 1978) that consolidated much of the work we had been doing. The thesis helped convince us that the time was appropriate for writing a book presenting our vision of an applied science of the user. Work on the book gave us the opportunity to focus on the larger vision instead of just the pieces, and we became aware of some missing unifying theoretical work that we felt we had to do.

Although we had several calculational models (Fitts's law for the mouse, GOMS models, the Keystroke-Level Model, plus other minor models), there were questions of relating these to the classical literature in cognitive psychology and human factors. We had long sought a unifying framework for tying together the relevant psychological knowledge about users (one of our 1974 working papers called this conception the "Handbook of Cognitive Man"), but had made little progress. In preparing the book we revived this goal and came to a much more satisfying conception, the Model Human Processor, which captured the relevant psychological literature in the terms of a unified, approximative model. This model also provided a foundation for our other models.

The other big missing piece for us was to understand the relationship between the cognitive skill we had discovered and the classical notion of problem solving in cognitive psychology. Here we built a theory of the behavioral continuum between problem solving and cognitive skill and showed how practice on a task would gradually change problem solving behavior into skilled behavior. This is an active area of research in cognitive psychology today (e.g., Anderson, 1981).

The result of these efforts was the book, *The Psychology of Human Computer Interaction* (Card, Moran, & Newell, 1983). However, the book represented only the main line of our research efforts that fit together into a tightly knit view. There were many other areas of our work that we decided not to put into the book, such as the issues of learning and of users' mental models.

4. The Conceptual Interface: Mental Models

The early effort of the AIP Project was focused on understanding expert user performance. As we explained in the previous section, skilled performance is characterized by methods that users know and quickly execute to

accomplish tasks. That is, we were focused on what is typically called procedural knowledge ("how-to-do-it" knowledge). However, from the beginning, we were aware that methods are not sequences of meaningless actions, but that expert users also have an understanding of what the procedures cause the machine to do. That is, the expert users have some sort of *mental model* of what is happening inside the computer ("how-it-works" knowledge).

The first AIP memo in 1974 proposed the notion of the "user's model," which refers to the conceptual model that the user can have of the system. A user's conceptual model is distinct from (but related to) the designer's implementation model. It is an abstraction of the system's architecture and software structures—the conceptual entities that the architecture and software implements—that is simple enough for non-technical users to grasp. (For example, a user might not know how the mechanism of the delete buffer of a text-editor works, but would simply know that the deleted text goes into a "clipboard.") A user's model would typically include knowledge of where information is stored (core memory, local disk files, remote file servers). It is important for the user to have an understanding of these kinds of features, for they are often not visible to the user.

The user's model provides an integrated package of knowledge that allows the user to predict what the system will do if certain commands are executed, to predict the state of the system after the commands have been executed, to plan methods for novel tasks, and to deal with odd error situations (by characterizing the system's state according to the model, then choosing operations necessary to leave that state).

Early Encounters with Conceptual Models

Early in our research we tried to write down rudimentary descriptions of the elements of conceptual models, which included the conceptual objects, their properties, how they related to each other (the characters in a text editor, for example, including the notion of a blank space as a character), and the conceptual operations that could be performed on those objects (inserting, moving, or deleting characters, including blank spaces).

Given the obvious importance of this kind of knowledge for the user, we were surprised to find that almost no system documentation ever clearly laid out a conceptual model of the system for the user. We were also surprised to discover what a difficult inductive task it is to specify such a model, even for a seemingly simple text editor. It was a detective game in which we were forced to hypothesize and test elements of possible models in order to find a succinct conceptual characterization of how the system worked. It was a game that took us days (not minutes) to do, hence not a game in which busy users were likely to engage.

It is clear that users attempt to make sense—by building mental models—of the behavior of a system as they use it. If a simple model is not explicitly or implicitly provided, users formulate their own myths about how the system works. These user-invented models may be inaccurate or misleading outside the very limited situation from which they emerged. Therefore, we believed that if the user is to understand the system, the system has to be designed with an *explicit* conceptual model that is easy enough for the user to learn. We call this the *intended user's model*, because it is the model the designer intends the user to learn. Just what mental model the user actually forms is another issue, which depends on how clearly the intended user's model is designed, how well it is implemented, and how well it is documented.

Although we were very concerned about the mental model issue, we didn't pursue explicit studies for several reasons: we didn't have a satisfactory methodology for studying it, we didn't have satisfactory representations of it, and we were busy pursuing the performance issues we have discussed. Still, we felt that the intended user's model was an important consideration in the user-interface design process.

User Interface Design Methodology

The practical application of our concerns came when Xerox began arranging the technology transfer between PARC and the Systems Development Division (SDD), which was created to develop office system products based on the research at PARC. In the Spring of 1976, a joint PARC/SDD committee (which included one of us) was formed to advise SDD on the design of the user interface of the office systems.

The committee decided not to try to design an actual interface, but to propose a methodology for SDD designers to follow in designing their interfaces. The methodology (Irby et al., 1977) included four parts: (1) analyze what tasks the user will want to do and the steps they go through to accomplish the tasks; (2) design an intended user's model in terms of which the tasks may be cast; (3) design a command language to make that model work; and (4) design an information display to reflect the operations of the system in terms of the conceptual model.

Thus, we recommended that the designer should lay out an intended user's model before designing the command language and the information display. The whole design effort should be oriented toward keeping this model "under control," i.e., keeping it simple, consistent, and clear enough for users to grasp.

The original designers of the Xerox Star interface, the workstation product SDD built, used this methodology. The conceptual model was clearly laid out in the system's functional specifications, and the designers worked hard to keep the model consistent. Although this model was represented informally, the fact that the designers focused on it contributed heavily to the widely recognized success of Star's user interface. The conceptual model is an under-appreciated aspect of Star's interface, but Star's more

widely-touted icons and desktop metaphor only make sense with respect to its underlying conceptual model.

William Newman, another member of the committee, presented this methodology in the second edition of his book with Bob Sproull (Newman & Sproull, 1979).

Empirical Studies

By 1979, we were ready to tackle some empirical studies in order to understand the role of the user's model. The first study was an attempt to elicit the knowledge that real users of real systems have about the systems they use every day. The system we chose to study was the Alto Executive, a system which was in wide use at PARC for several years by nontechnical support people as well as programmers.

We wanted to find out what Alto users knew about the Executive. The goal was to see whether we could find some kind of mental models buried in the user's knowledge. The methodology we used was to transcribe interviews with several users into logical propositions and then classify them into categories.

The surprising result was that many nontechnical expert users (e.g., secretaries who used the system effectively every day) gave no evidence of having anything but very shallow models of how the system worked. (Perhaps we shouldn't have been surprised, because no conceptual model was documented and training was informal.) This led us to consider more carefully the role of the user's model.

In order to characterize the role a user's mental model would play in the use of a system, we performed experiments with the simplest kind of system we could devise, a simple stack-based calculator (Halasz & Moran, 1983; Halasz, 1984). We thought that a model of a stack might well be useful to help rationalize what for many users is a nonintuitive postfix command language. The formal experiment compared one group of users who were taught an explicit conceptual model of the stack, and a group of users who were carefully shielded from the stack model. The model group was trained in relation to a specific model and were told how that model related to the methods for solving arithmetic problems. The no-model group was only taught specific methods for performing the same tasks. Then we gave the two groups sets of problems that were categorized as simple routine tasks, slightly more complex tasks, and very difficult "invention tasks" (which required the user to invent new methods to solve).

The results (see Figure 11) revealed no difference between the two groups in both the routine and complex problems, but the model group performed much better on the invention tasks. The most surprising result was that even some of the no-model group were able to perform some of the invention tasks. We wanted to account for these results according to the cognitive theories we understood, so we analyzed the protocols gathered from the users as they performed the tasks.

Problem Type	No-Model Users	Model Users
Routine	98	95
Complex	87	88
Invention	25	67

Figure 11. Percentage of problems correctly done in the calculator experiments. There were two groups of users; one was taught an explicit conceptual model of the calculator's stack and the other group was prevented from having such a model. There were three types of problems, each of a different difficulty relative to what the users were taught.

Problem Type	No-Model Users				Model Users			
	Skilled Method Execution	Problem Solving			Skilled Method Execution	Problem Solving		
		Model Space	Methods Space	Task Space		Model Space	Methods Space	Task Space
Routine	90	0	8	2	89	11	0	0
Complex	94	0	6	0	91	7	0	2
Invention	2	0	84	14	7	71	19	3

Figure 12. Partitioning of the users' behavior in the calculator experiments. The users' behavior was devided into four behavioral modes: skilled method execution plus three problem solving modes, which are distinguished according to which problem space they were working in. The partitioning is based on an analysis of verbal protocol records; each line of protocol was categorized into the mode it manifested. The numbers in the table are the percentages of lines of protocol in each behavior mode. We believe that this measure underestimates the amount of work in the task space.

We divided the users' behavior into skilled method execution, as we had modeled in our earlier studies, and problem-solving. We found (see Figure 12) that in the routine and complex problems, the behavior was almost all skilled method execution: the subjects had been taught what to do, and they did it; even for the complex problems it wasn't difficult to knit together the methods they had learned for solving the problems.

Our hypothesis going into the study was that the conceptual model we taught the model group of users would provide them with a problem space through which they could search in order to find solutions. According to this theory, the user would characterize a difficult "invention" task in terms of this problem space: the state of the system when they started, the state of the system they would like to achieve, and a set of operations to move them through that space; they would solve the problem using generic problem-solving strategies.

However, when we analyzed the users protocols, we found two other kinds of problem spaces in which the users worked when they were problem solving. One was a *task space* within which they manipulated the given arithmetic task in various ways, such as dividing it into subtasks that could be solved by known methods. Another space in which they worked was a *methods space*, where they took known solution methods as strings of steps, manipulated those steps in various ways to produce new methods, and tried them to see how they worked.

We found that the most critical (although not very time consuming) problem solving was in the task space, where the user analyzed the given task into subtasks and delegated them to the model space or the methods space. The main difference between the users who had a model and those who didn't was that they had different problem spaces in which to work (see Figure 12). The model space was an effective problem space, within which the solution to the invention problems could be found; the method space was not particularly effective, but it was sufficient to allow some non-model users to stumble onto solutions to some of the invention problems (often, much to their surprise).

We concluded that mental models can be useful for novel task situations, but we found that users only use their models in specific subtasks; there was a lot of switching between the task space and other problem spaces. Users were cautious about going into a mode of behavior that involved thinking through a mental model. Model-based problem solving appears to be very mentally intensive, so users avoid it if they can apply cognitive skills. But, if users don't have appropriate methods available, they will retreat to some kind of problem solving. In these cases, a good conceptual model provides an effective problem space in which to work.

Thus, system designers should think of a conceptual model of a system as not just a simple view of a complex system, but as a problem space through which users can search for solutions to a variety of novel problems. The conceptual entities and operators in the intended user's model should be closely related to the kinds of tasks the users are likely to do, and the users should be provided with heuristics for moving through the model space.

Theoretical Studies: Task Mapping

We also worked on a theoretical analysis of conceptual models to show where they fit into the overall structure of the user interface. The Command Language Grammar formalism (Moran, 1978, 1981) shows how models relate to the task domain, the command language, and the detailed user-computer interactions. According to this theory, the conceptual model provides the user with a link between his task domain and the syntax of the interactive dialogue. That is, on the one hand the conceptual model serves as the semantics of the dialogue actions, while on the other hand it serves as a base into which the task can be mapped into the system.

Richard Young called this kind of mechanistic conceptual model a *surrogate model* (Young, 1983). Interactions on these issues with Young, who was exploring the domain of simple calculators, led to the discovery of a new kind of mental model—*task-action mappings* (Young, 1981). The properties of radically different calculator designs, such as algebraic versus stack calculators, could be best understood by an analysis of how well calculation tasks could be directly mapped into the actions available on the calculators. Surrogate models were completely bypassed in this analysis, which helps explain why people sometimes seem to get along with systems without having surrogate models of them.

In further work along this line, we have proposed a calculus, called *ETIT analysis*, for task mapping (Moran, 1983). The "fit" of a system to a task domain can be assessed by enumerating rules for reformulating system-independent task descriptions ("external tasks") into system-specific task descriptions ("internal tasks"). Rule-based system description techniques, such as this or Payne's (1984) *task-action grammar*, are beginning to provide a way to help us formalize the fuzzy notion of the *consistency* of a system (both internal consistency and consistency with respect to a task domain). Such techniques look promising as a way provide system designers with calculational techniques for predicting the learnability and "guessability" of systems.

5. The Task Interface: Pondering Ideas

Let us now turn to the role of a science of the user in the future development of the personal workstation. For us, the real challenges for user technology are now at the larger task level in which users are grappling with complex intellectual tasks. This is, of course, a return to working directly on a modern version of the original vision of augmentation workstations.

We now have workstations powerful enough to give each user a personal system equivalent to one that only a decade ago would have been shared with a hundred other users, and we are on the verge of major upgrades to even greater computing and communication power. We now have a much more developed base for user technology—not only techniques for designing user interfaces, but also models for understanding users. Thus, we are now in a better position than ever before to explore systems that can really augment human intellectual endeavors.

The challenge is to create systems that, through intimate cognitive interaction with users, aid them in structuring and manipulating their ideas. With such systems to help them, people will then be able to cope with more and more complex intellectual tasks of all sorts—authoring books and multi-media presentations, designing products and programs, composing music, analyzing experiments, the arguing points of law and policy, reasoning about scientific and social issues, and on and on.

The key to building such systems is to find ways by which a user can act on his ideas as objects, just as current text editors allow him to act on words as objects. This is difficult, since ideas are often tacit and ill-formed. A means is needed to *externalize* the ideas, to get them out of the user's head and into a form that can be organized and shaped. Two problems require solution:

(1) The user needs new ways to *represent* his ideas—to get the user's mind around the ideas, as it were. For this we can exploit advances in artificial intelligence and cognitive science.

(2) The user needs new ways to *manipulate* his ideas—to get the user's hands on the ideas. For this we can exploit advances in interactive computer graphics.

These problems are part of our current research agenda. In this endeavor we must not only build on our current base of understanding of the user, but also advance that understanding. For one thing, we have to launch studies into the nature of the complex intellectual tasks we wish to augment. This calls for a shift in our research strategy from studying users of existing systems to studying users of new systems that we ourselves build, which enables us to understand the nature of the tasks and the limits of users and systems for dealing with them. We give illustrations of this strategy from some of our own current projects.

Representation: Idea Structuring

Word processors and text editors, even powerful ones, are tailored for the final stages of writing a paper—crafting the text and graphics of the final product. Outline processors help with the previous stage of outlining. Although the latter are sometimes called "idea processors," it is clear that idea processing begins well before the stage of outlining. A genuine idea processor should allow the user to deal with ideas that are vague and ill-structured and help him gradually add structure as it is discovered.

We use the term *authoring* to refer to the larger intellectual task of gathering information, extracting and discovering ideas, structuring them, and finally composing them into a readable product. Authoring in this sense is a highly general task composed of generic subtasks, as illustrated in Figure 13. One begins by collecting sources of information; from these a set of notes (idea-sized units) are created to represent potentially relevant facts and ideas. As notes accumulate, they need to be filed in a structure suitable for retrieval, such as a topic hierarchy. But these notes also need structures that organize the ideas into meaningful, coherent themes. Such structures must be discovered, elaborated, supported by evidence in the notes and other sources. One then communicates these ideas by composing them into an interpretable product—a document, a slide presentation, or a browsable network of ideas.

The key research issue here is to help the users develop explicit mental models of idea structures, so they can see them, play with them, and evaluate them. This requires the invention of representations for *externalizing* ideas and idea structures. Cognitively, these representations serve the

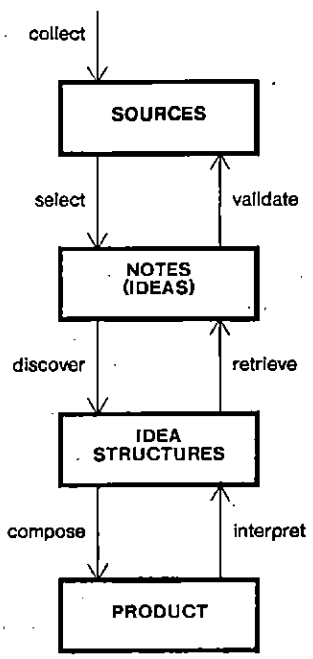


Figure 13. Schematic diagram of the generic authoring task. The boxes are the types of information to be managed in the authoring process, and the arrows are some of the processes for generating and transforming the information.

users as problem spaces; and new representations can fundamentally alter the way they are able to deal with ideas. Some representations should be "weak" in the sense that they can easily capture a large variety of even vague ideas; other representations should be "strong" in the sense that they can precisely capture and process particular classes of idea structures; and facilities must be provided to help users transform weak representations into strong ones.

We are developing a system, called *NoteCards*, to explore the broad nature of the authoring task. NoteCards supports the orienting metaphor of the notecard as a medium for capturing an idea. Notecards can be stored in fileboxes and linked into complex structures, which can then be viewed in graphic representations (such as the one shown in Figure 14). NoteCards is designed to be an open, flexible environment, so that we can use it as a "laboratory" to explore new representations and tools to support authoring tasks. Because it is an open system, NoteCards users are faced with the problem of devising appropriate ways to use the system, that is, mapping their particular tasks onto the basic elements of the system. This leads to a further interface issue of making the environment tailorable by users themselves to the wide variety of situations that we encourage them to bring to NoteCards (paper and documentation writing, scientific and legal argumentation, instruction authoring, design analysis). Thus, in contrast to our earlier methodology of observing a controlled, skilled user population, we are now observing a population of idiosyncratic, exploratory users.

Manipulation: Idea Browsing

The representations in an idea-structuring environment must be assimilated and manipulated with facility and speed. Idea-processing tasks are difficult for users, because they involve the retrieval, tracking, and manipulation of a large number of ideas (as is being attempted in Figure 14). Authoring a paper, for example, may require hundreds of notes and scores of references; programming may require hundreds of routines. But, as the Model Human Processor

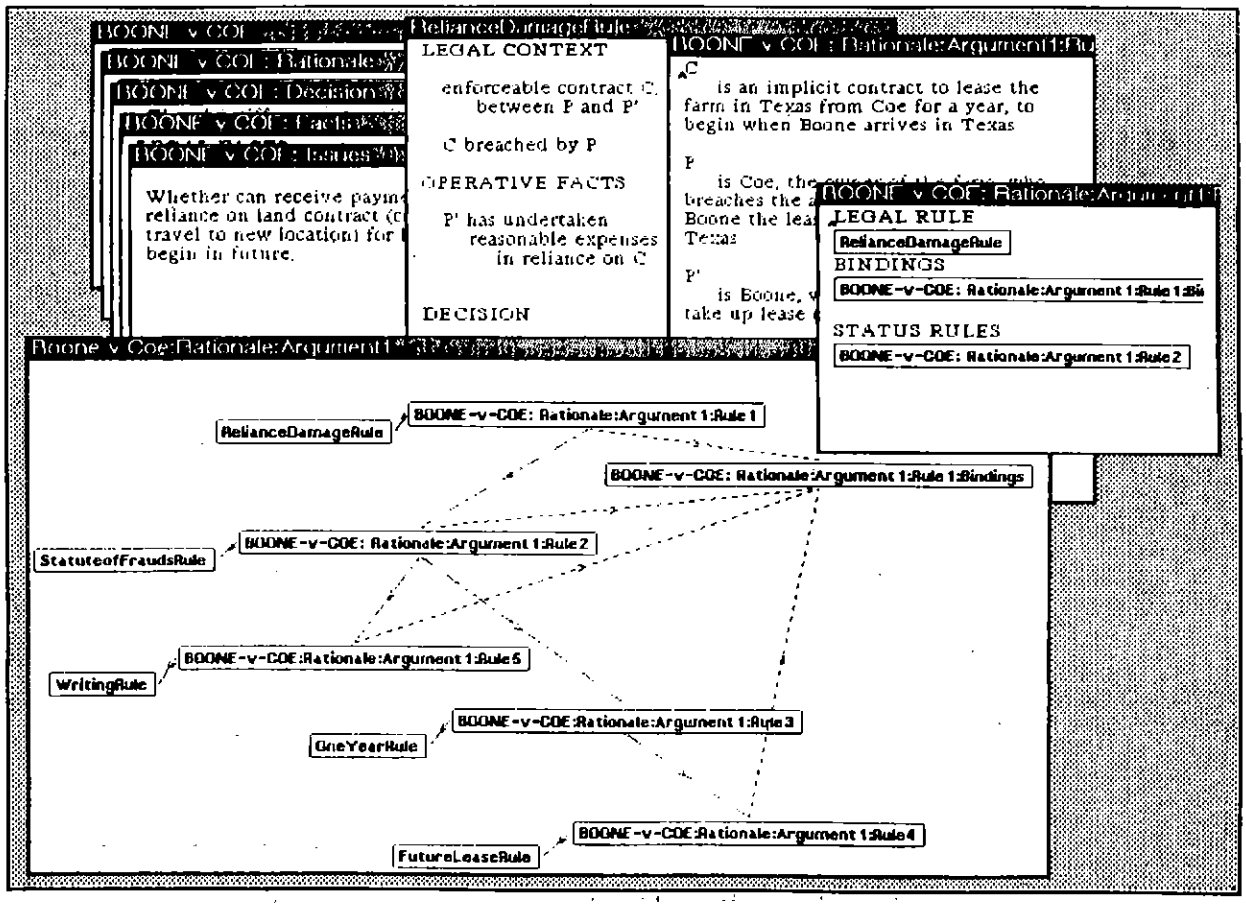


Figure 14. A screen image of the NoteCards system. The application shown is the analysis of a legal argument. A variety of legal rules must be brought to bear in arguing a particular case. The graph shows how the rules are related by applicability conditions and by bindings to the particulars of the case.

shows, only limited amounts of information can be handled in the user's working memory. These limitations lead the user to structure his behavior into unit-task bursts.

These considerations lead us to ask how a computer system can compensate for human cognitive limitations. The display can be used as an external memory to augment the user's internal working memory. This partly explains what has made the "desk-top" metaphor successful: users can use windows and icons to keep track of more documents, notes, and messages from other users than with the previous style of command language interfaces. By having these things visible as reminders, the user's effective working memory is increased, enabling him to do more complex tasks.

But the problem is that these techniques, which work well with with a few dozen objects, do not scale well up to the several hundred or even thousands of objects necessary for the idea-processing tasks we wish to do. Here, it helps to begin with a better understanding of the nature of the problem. Again, our method for gaining that understanding is through approximative models of user behavior. In this case, we have sketched a model called the Window Working Set Model (Card, Pavel, & Farrell, 1984) that analyzes access to screen objects in a manner analogous to the analysis of virtual memory operating systems. Informally, this model suggests that screen space itself is the key constraint and that at some point as the number of, say, overlapped windows required increases, user performance will decrease in a sudden and non-linear way, sending the user into the window version of thrashing.

Thus there are limitations on the uses of current graphical interface techniques for browsing large sets of ideas. Advanced graphics systems, however, open up a new set of possibilities. Figure 15 shows a fragment of an

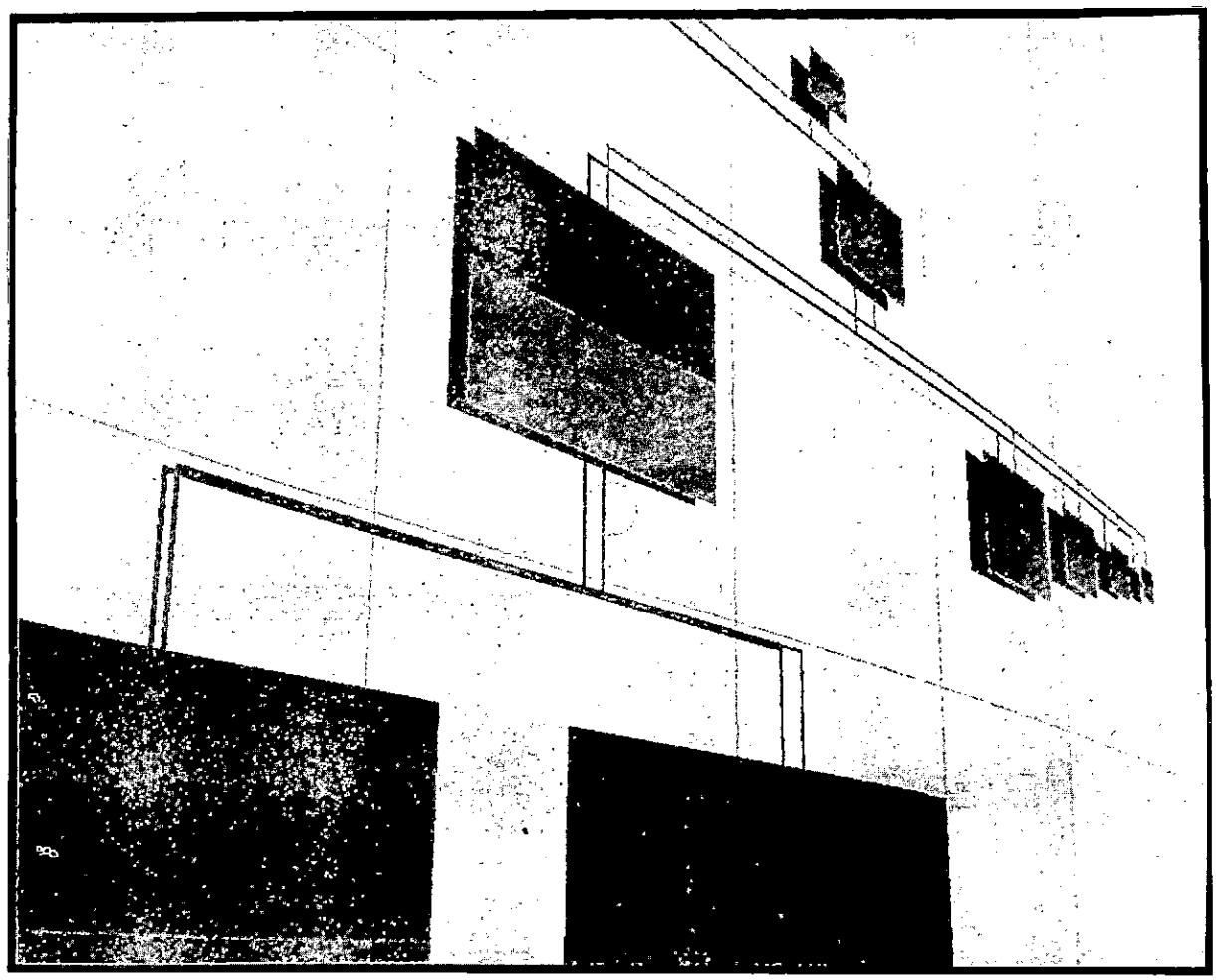


Figure 15. Experimental Dandelris browser for organization charts. The photograph shows a view of a fragment of an organization chart as seen by a user flying around this chart. The chart is laid out on a simulated 2-dimensional whiteboard. The user can change his location in simulated 3-dimensional space, his direction of gaze, and by pointing with the mouse, to indicate focus, the relative sizes of different organization chart boxes. (The experimental program was written by S. Card, A. Henderson, L. Lovstrand, and B. Verner.)

organization chart from an experimental browsing system we are implementing (the DandeIris, combining a Xerox DandeTiger Lisp machine and a Silicon Graphics Iris graphics processor). The node that is the user's focus is largest and is readable in the most detail. Nodes become smaller with distance from the focus. The user can fly around this chart to see it from different points of view and change his direction of gaze; if he points at a node, indicating a new focus of attention, it grows larger. The transitions are all animated in real-time, color, three-dimensional representations to help the user keep track of the transformed identities of various objects.

Such a display is designed to help the user navigate among a large number of objects; only a few of these objects would be visible in detail at any time, but many more would be visible as orienting cues and retrieval keys. The visual movement techniques is coupled with other retrieval techniques for allowing the user to focus on a limited number of items at a time (as required by our understanding of user's processing capacity) while retaining rapid access to a large number of items (as required by our understanding of the requirements of complex intellectual tasks). By coupling experimental programming with the analysis of user behavior, we hope to find theory-based techniques to aid the user in keeping track of and maintaining effective browsing and retrieval capability for large information structures.

6. Conclusions

What has been learned about users and how does this relate to the original visions of personal workstations? The short answer is that, unlike the early 1970's when little was known scientifically about computer users, we now have a vision of the form of an applied science of the user and a few areas of knowledge where that vision has been realized. In this paper we have narrated some of our efforts at building this applied science:

At the physical interface level, we have discovered that user performance with pointing devices is constrained by the information-processing capacity of the user. We have learned the quantitative law describing this constraint and have determined that certain devices, such as the mouse, are at the performance limits allowed by this law.

At the cognitive level, we have learned that routine human-computer interaction, such as text-editing, does not involve problem solving, but rather cognitive skills based on the execution of known methods. We can see the information-processing constraints of the user show through this skilled performance in the unit task of users. We have characterized cognitive skill to the extent of developing an engineering model for use by user-interface designers.

At the conceptual level, we have learned that users often have mental models of the systems they use, and that such models enable performance of novel tasks. System conceptual models provide the basis for users to acquire mental models, and thus are an important basis for system design. But mental models are cognitively intensive, and users will avoid them by attempting to map directly from their tasks to the actions required in a system. Theories of task mapping are just beginning to emerge.

At each of the above levels, it has been possible to base an applied science on a theory of the cognitive mechanisms underlying user behavior. At each level, the applied science was shown to be practical, in particular by being influential in the design of the Xerox Star.

The challenge for us today is to use our understanding of users to discover new ways to augment users in complex intellectual endeavors. Thus, we are concentrating our efforts in those areas where users' cognitive limitations need to be overcome and where users' cognitive abilities can be aided with computer-based tools. We believe that the most interesting problems are at the task level: understanding the nature of complex intellectual tasks and finding ways to build idea-structuring tools, both representation tools for structuring ideas and display tools for browsing ideas.

Whereas we once studied users empirically using existing systems in order to understand the nature of human-computer interaction, we now use what we have learned about users to help drive the creation of new experimental systems.

REFERENCES

Anderson, J. R. (1981). *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Bell, C. G., & Newell, A. (1971). *Computer structures: readings and examples.* New York: McGraw-Hill.

Bush, V. (1945). As we may think. *The Atlantic Monthly,* August.

Card, S. K. (1978). Studies in the psychology of computer text editing systems. Unpublished doctoral dissertation, Carnegie-Mellon University, Department of Psychology.

Card, S. K., English, W. K., & Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics, 21,* 601-613.

Card, S. K., Moran, T. P., & Newell, A. (1976). *The manuscript editing task: A routine cognitive skill* (Technical Report SSL-76-8). Palo Alto, CA: Xerox Palo Alto Research Center.

Card, S. K., Moran, T. P., & Newell, A. (1980a). Computer text-editing: an information-processing analysis of a routine cognitive skill. *Cognitive Psychology, 12,* 32-74.

Card, S. K., Moran, T. P., & Newell, A. (1980b). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM 23,* 396-410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Card, S. K., Pavel, M., & Farrell, J. E.. (1984). Window-based computer dialogues. *Proceedings of IFIP Interact '84,* 355-359. London: IFIP.

Engelbart, D. (1963). A conceptual framework for the augmentation of man's intellect. In P. W. Howerton and D. C. Weeks (Eds.), *Vistas in Information Handling,* Vol. 1. Washington, D.C.: Spartan Books.

English, W. K., Engelbart, D. C., & Berman, M. A. (1967). Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics, HFE-8,* 5-15.

Halasz, F. G. (1984). *Mental models and problem solving in using a calculator.* Unpublished Ph.D. Dissertation. Stanford, CA: Stanford University.

Halasz, F. G., & Moran, T. P. (1983). Mental models and problem solving in using a calculator. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Irby, C., Bergsteinsson, L., Moran, T. P., Newman, W., & Tesler, L. (1977). *A methodology for user interface design.* Palo Alto, CA: Xerox Palo Alto Research Center.

Kay, A. (1977). Microelectronics and the personal computer. *Scientific American,* September, 230-244.

Licklider, J. C. R. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics, HFE-1* (March), 4-11.

Moran, T. P. (1978). *Introduction to the Command Language Grammar* (Technical Report SSL-78-3). Palo Alto, CA: Xerox Palo Alto Research Center.

Moran, T. P. (1981). The command language grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies, 15,* 3-50.

Moran, T. P. (1983). Getting into a system: external-internal task mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Newman, W., & Sproull, R. (1979). *Principles of interactive computer graphics,* 2nd ed. New York: McGraw-Hill.

Payne, S. J. (1984). Task-action grammars. *Proceedings of Interact '84: First IFIP Conference on Human-Computer Interaction,* London. Amsterdam: Elsevier.

Young, R. M. (1981). The machine inside the machine: users' models of pocket calculators. *International Journal of Man-Machine Studies, 15,* 51-86.

Young, R. M. (1983). Surrogates and mappings: two kinds of conceptual models for interactive devices. In D. Gentner & A. L. Stevens (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum Associates.

NOTES

NOTES

Conference Proceedings

ACM Conference on the History of Personal Workstations

Palo Alto, California
January 9-10, 1986