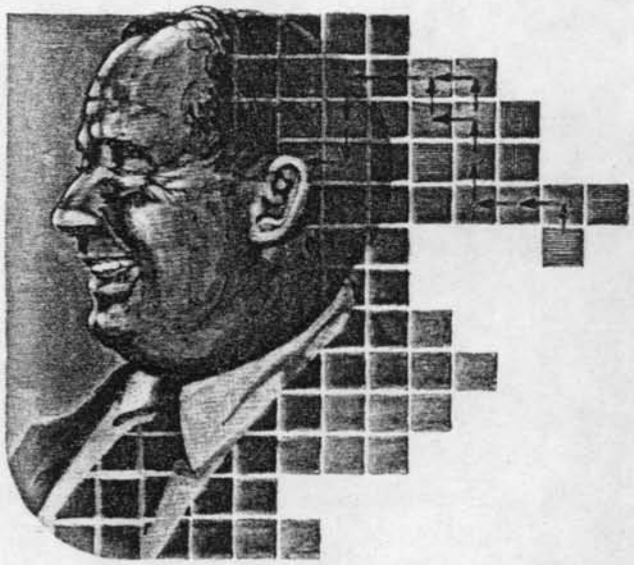
u2797

The Evolution of Software


John von Neumann

INTRODUCTION

In less than three decades, software technology has left its imprint upon mankind. A whole new field of learning has been opened. Data processing has unalterably changed government, banking, industry, public utilities, transportation, commerce, etc. There has even been a revolution in language as new words and phrases become part of everyday speech and old words — such as hardware and language — take on new meanings.

Software development has been largely based on expedience and demand. Throughout this evolution there have been time lags — hardware for which there was no software or, sometimes, software concepts that were beyond the capability of the machine. Neither, however, is successful without the other.

Computer software received a great stimulus when one of the world's leading mathematicians, Dr. John von Neumann, came up with the concept of a permanent all-purpose memory in his draft report on a new computer . . . the EDVAC or Electronic Discrete Variable Computer . . . introducing computer instructions, or subroutine series, for the performance of specific operations. Subroutines could be used over and over again by any other program that required the same type of operation. Thus, the tentative and limited start of programs — software as opposed to the hardware or machines on which they operated.

SCIENTIFIC AMERICAN

The 1890 census was compiled by means of Herman Hollerith's electrical tabulating machine.

In May 1949 at Cambridge University in England, the EDSAC — Electronic Delay Storage Automatic Calculator — performed its first computations, the first performed by a stored program anywhere. This was soon followed by an American stored-program computer, the National Bureau of Standards Eastern Automatic Computer — or SEAC — placed in operation in 1950.

The SEAC computer was designed to be an experimental computer to test different types of input/output and computer memory. It started life with 512 words of mercury delay line memory and an input/output teletype. A floating point interpreter for the SEAC — eventually to account for over 50 percent of its production time — was developed by Dr. Charles Swift, a senior scientist on CSC's staff. SEAC is credited with being the first machine designed to operate on its own instructions that was able to sustain full steady operation.

Up to now, the computer was taking form in obscure academic laboratories encouraged by funding from government, military, and research organizations. In 1951, the U.S. Bureau of Census selected the Univac I — the first large-scale electronic computer available for business use — to help process its massive burden of data. As an interesting sidelight, the U.S. Bureau of Census was the same agency that had started Hollerith on his punch-card equipment in the 1880s. Many new features were introduced by the Univac I: the use of magnetic tape for external data storage, a programmed system to sort and merge records without first having to read them all into the computer's memory, and an editing system that was the forerunner of many modern report generators.

Although by the early fifties the computer concept was firmly established, programming was still at a primitive stage. Programs had to be written in machine language, a painstaking linking of lists of numbers, letters, and special characters identifying the location of each instruction and item of data in the computer's memory. The software problem was still to be attacked.

The software problem involved more than a search for people to perform routine procedures. The technology was new. There were no established pathways or precedents. There was a need for a singular kind of talent: people with the technical background to understand computer architecture, and the strategic and innovative abilities of the creative. With the exception of government agencies and the very large manufacturers, virtually no organizations existed that were capable of generating computer instructions — software — well enough or fast enough to meet the demands of the new large-scale machine.

It was estimated that if the exponential growth of computers, software, and programmers continued at today's rate, it would require more than one-half million people by the end of the 1970s. This adds up to ten percent of Americans between 25 to 45 who are able to learn to program. This is not unlike the telephone growth analogy. If the telephone company had not gone to dial telephones to keep pace with demands, today every woman between the ages of 20 to 50 would have had to be a telephone operator.

Obviously as more and more machines became available, the pressure increased to find ways to simplify instructions to the computer and improve the productivity of people. A new type of resource emerged, the software house that could economically maintain a staff of computer scientists on a continuous basis. Stimulated by the cross-fertilization of human talents and machine concepts, computer scientists are playing a significant and vital role in the growth of information technologies.

sharing

One of the most significant characteristics of early software development was the nationwide sharing of ideas, developments, and completed program products. Against the backdrop of World War II, there was a sense of sharing that might perhaps not have happened in peacetime.

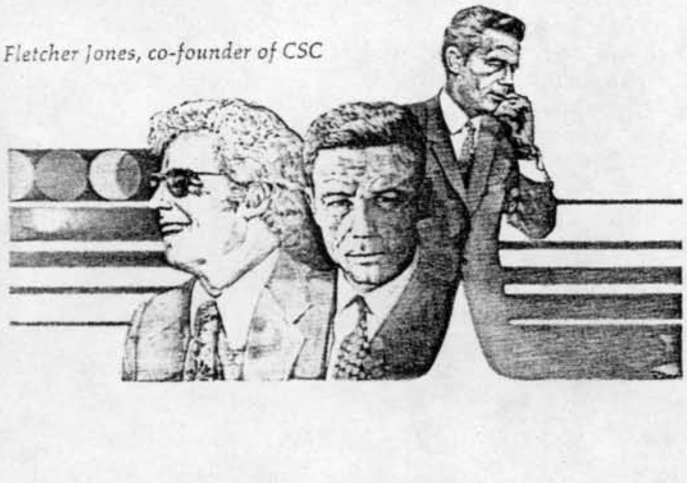
One of the earliest attempts to organize and standardize software practices resulted from the friendly interchange of information among scientific users of the IBM 701. Eventually, this interchange was formalized into the PACT Committee with the declared aim of developing an operating compiler. Unfortunately, the computer was obsolete before the PACT compiler was perfected. PACT was most significant for its effect on later developments, such as SHARE, and the concepts carried on by its members.

The SHARE committee was formed shortly after the announcement of the IBM 704 with the primary aim of sharing information, experiences, and new discoveries to avoid duplication of efforts.

For the most part, sponsored by defense industry companies, SHARE initiated a tradition of free and timely communication between cooperating user groups. Since its primary aim was to coordinate software development for all users of IBM machines, it served to attract some of the key people in the growing data processing industry.

Accounts of the early proceedings of SHARE were covered by an informal newsletter, which was assembled by Fletcher Jones, the organization's first secretary and, later a co-founder of Computer Sciences Corporation. Membership was by institutions, with key defense companies and government agencies represented. Of the attendees at this first meeting, six of them were to become members of Computer Sciences Corporation.

Fletcher Jones, co-founder of CSC



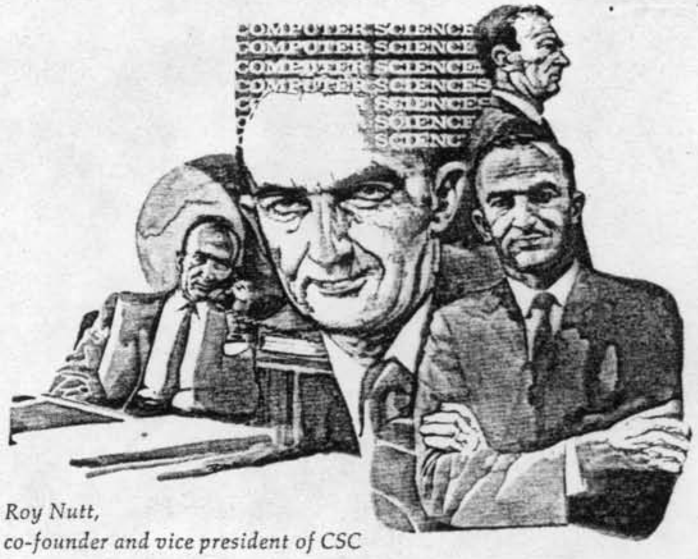
Computer Languages

One of the most important products of disciplined habits of inquiry and analysis was the software that converted human ideas to machine language. These language processors include such systems as assemblers, compilers, and interpreters, each with its own set of advantages and disadvantages.

Computer languages, like human languages, run the whole gamut of sophistication, from machine-level coding systems using strings of numbers (which can be compared to primitive sign language), to high-level interactive language, using grammatical syntax, and English words and statements.

assemblers — the first step

If for no other reason than the fact that they are the earliest and by far the largest class of language processors, assemblers deserve first coverage. Assembly language is closest to the language of the machine. Basically, the assembler converts the programmer's symbolic form instructions into the machine language that the computer understands. As a general rule, each line of code in an assembly language program is associated with a single machine operation. Thus, an assembler is generally designed for use on one type of computer only and the language used is meaningless with reference to any other machine. Often, machines within the same family cannot interchange assemblers. Such specialization, on the other hand, makes an assembler an effective way to access all the features of the machine, and consequently, the principal tool of software architects — the systems programmers. Good assembly code executes faster and is more compact than code written in any other language.



Roy Nutt,
co-founder and vice president of CSC

Early assemblers did very little beyond "relative" or "regional" addressing. They contained no subroutine libraries, no notions of "location counter," no symbols, and no listing control. Modern day assemblers are much more sophisticated, but each step toward sophistication takes the program writer one move farther from the machine.

From the halting step-by-step procedures performed by assemblers, programmers began to search for methods that would reduce, or optimize, machine time in accessing instructions or data. Some early examples of optimization by an assembly program were developed for Univac 1 in 1951 and, subsequently, for the IBM 650 in 1952.

Up to this point, systems programmers had been busily engaged in developing individual assembler systems not only for each different machine but often for each different user installation — all of which made language processor standardization an essential goal for increased computer usage.

In 1956 the race for a standard assembly language to be used by the 704 became a five-entry affair with two sections of IBM hotly contesting each other. Elaine Boehm's "regional assembler" (IBM Engineering) had mnemonic operation codes and a Dewey-decimal type of addressing system. John Greenstadt's NYSAP entry (IBM Marketing) program was a fixed-field symbolic assembler. Lou Gatt at Los Alamos Scientific Laboratories came up with a variable field, symbolic assembler with non-IBM mnemonics. The CAGE entry from Donald Shell at General Electric featured a free-field symbolic assembler with addresses and special symbol definitions. The standard chosen was UASAP — a symbolic assembly program with extensive symbol arithmetic — developed by Roy Nutt, vice president and co-founder of CSC, then at the United Aircraft Research Computation Laboratory.

UASAP later became known as SAP for Share Assembly Program. It did little more than one-to-one translation and library inclusion, and the programmer had to control the parameters of the program. However, as basic as SAP was, it did offer more than its predecessors. Its principal contribution to the software mainstream was the capability of programming addresses as a combination of symbols and decimal integers.

Autocoder, which was a sort of assembler with convenient macro-instruction libraries, was produced in 1955 for the IBM 702/705 series. It was able to hold the variation from one machine to another to a minimum, which evidenced some achievement for the organizations laboring for software standardization.

The prevailing theme in software is man's service to the machine — with an obvious slant toward man. The machine itself, of course, is actually the result of human efforts, but several steps removed from the final product; thus, the individual's contribution is less evident in terms of singular achievements. Software, on the other hand, is the direct result of an individual, or at times, small groups of people.

Man's service to the machine sometimes is Herculean. Typically, there's the story of software for the Univac III. It's only one of several that could be told about the achievements and strategies of highly skilled professionals in producing software systems. In this case, the prodding need was software for the new computer, which was ready for delivery but lacked satisfactory software. Due to unusual circumstances only two days could be spared — over a weekend — on the prototype machine. Three CSC programmers — Roy Nutt, Owen Mock, and Dave Ferguson — condensed months of preparation into three to four weeks of intensive work prior to taking off for Philadelphia where the computer was still being tested.

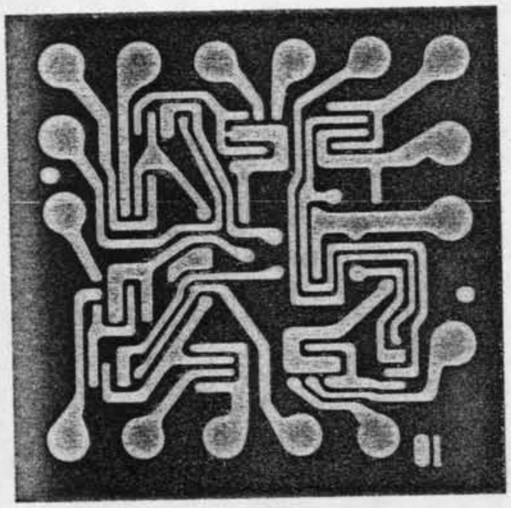
After a crammed weekend of debugging, the team emerged with an interim assembly program, an octal loader, a relocating loader, a dump program and a rudimentary executive program. Enough to get Univac III off the ground.

interpreters

The interpreter does not produce a program for later execution, but instead translates and executes "on the fly," that is, it translates programmer's code as the computation progresses and performs required operations through subroutines.

Interpretive systems appeared to be the way to go in 1952 when IBM introduced its large-scale 701 binary computer, and Speedcode—a mathematically oriented language—demonstrated how far overburdened programmers were willing to go in sacrificing computer speed for the sake of programming convenience.

Larger memories, such as that of the IBM 704 in 1956, with 4096 words of magnetic core storage, made interpretive systems like Speedcode with lengthy program running times less attractive.



compilers

Users with a particular application needed a relatively simple and easy-to-learn language that would give them direct access to the computer without a lot of detailed instructions that cost time and increased the possibility of errors.

By 1957, there was a hopeful new approach to more powerful software — the compiler. Where an assembler translates item for item into machine code, the compiler translates problem-oriented language directly to machine code.

FORTRAN — FORmula TRANslation — was the first successful algebraic compiler. It was the first compiler to effectively optimize the use of index registers to produce code without unacceptable degradation of performance.

The guiding light in FORTRAN's development was John Backus. Starting in 1954, Backus, with the strong support of IBM Vice President John McPherson, spent some 25 man-years in developing the first workable compiler.

Several corporations and research organizations pooled their brainpower to support the IBM project. For example, United Aircraft dropped its own algebraic compiler and lent Roy Nutt to IBM "half-time" for more than two years. Sheldon Best left MIT to participate in the project.

Working conditions did not match the talents of the assembled team. The project was carried out in a "grundgy" building near the IBM World Trade Center in New York City. Machine time was so scarce and unreliable that the development team maintained a 24-hour watch on the computer, usually by sleeping in the computer room!

Early FORTRAN compilations were slow, but the compiler introduced a number of ingenious features to produce computer routines comparable to those produced by an average programmer writing directly in machine code. Many FORTRAN concepts of optimization and code generation overshadowed the field for the next decade.

The first FORTRAN was a manager's nightmare. It was written in a variety of languages, partly in UASAP, partly in NYSAP, and partly in a special machine language for a loader that could accept octal (base eight numbering system) addresses and mnemonic operation codes. Most of the NYSAP code was never reassembled but patched and repatched until the compiler was running properly.

When IBM introduced the 705, the commercially-oriented users formed their own group similar to SHARE — GUIDE — which planned to expand FORTRAN for the new machine. IBM didn't agree, so GUIDE, chaired by James Matheny (subsequently, a senior member of Computer Sciences Corporation's original technical staff) went off on its own and wrote a second FORTRAN compiler — however with heavy assistance from IBM. The result was FORTRAN II with subroutines, a major breakthrough for building large programs without excessive compiling time. While it didn't represent a big change, user subroutines could be compiled and debugged separately and then plugged into a main-line program. By 1960 others were ready to join the FORTRAN club: Univac, Philco, Honeywell, Control Data, and Bendix soon qualified with units that accepted FORTRAN source programs.

As computer equipment advanced, each new version of FORTRAN was able to take advantage of its hardware. Common agreement held that one of the best FORTRAN compilers ever produced was designed by Dr. Gordon Rice and Sheldon Sidrane, both senior scientists on the CSC staff in 1964 for the Philco 2000. Its advantage was a capability to use some unusual instructions and produce object code an order of magnitude better than the compiler it replaced, demonstrating at the same time the feasibility of fast compilation and exceptional efficiency. FORTRAN IV for the Univac 1107 extended the language to include bit and character manipulations, use of remote terminals, added Boolean operations, and expression substitutions similar to assembly language macro-instructions.

After delivery of FORTRAN IV to Univac in 1964, CSC developed added extensions to this compiler for in-house use, and the compiler, renamed FORTRAN V, later became part of Univac's EXEC 8 operating system as well as CSC's own timesharing system.

the computer learns english

While FORTRAN simplified programming tasks for scientific users, it didn't fulfill the needs of businessmen, who use their computers primarily to process data, rather than perform mathematical computations. The object of business users is to acquire output reports by either sorting, converting, or editing quantities of input data.

Early in the history of software, a special programming research group was formed under Dr. Grace Hopper, often referred to as "the mother of software," by the Eckert-Mauchly Computer Corporation to pursue her concepts of a "problem-oriented" language that would allow problems to be stated in simple, English-like terms without having to specify each detail of machine operation.

Dr. Grace Hopper

Dr. Hopper and her staff made a significant contribution and one that was to have considerable influence on later business-oriented languages when they developed FlowMatic, the first major processor to use English language words and natural grammar.

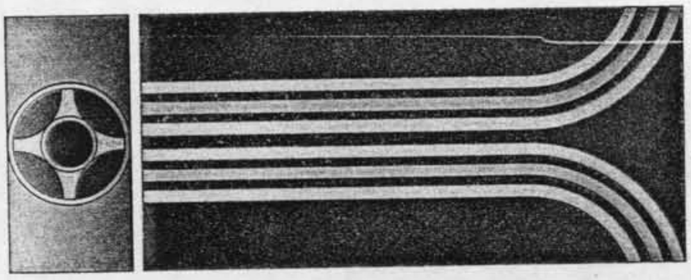
In 1959, the Datamatic Division of Minneapolis-Honeywell Regulator Company contracted with the newly-formed Computer Sciences Corporation for a business compiler for their Honeywell 800 machine. Named FACT, for Fully Automatic Compiling Technique, it was a fully integrated, problem-oriented, commercial language.

FACT went to great lengths to permit the use of "natural" language. For instance, the programmer could write "2 TIMES A," or "2 MULTIPLIED BY A," or "2 * A" and the computer would accept any variation. Several features of FACT, which were novel at that time, have since reappeared — character conversion and automatic explosion/implosion of file data.

FACT was also one of the first systems that dynamically allocated storage and allowed portions of a record to be stored in nonresident form. It contained a report-writer that was years ahead of its time, an integral SORT verb and an UPDATE verb that preceded functions later to be supplied by data management systems.

At the same time that FACT was in development, representatives of the major manufacturers and users of data processing equipment were meeting in Washington at the urging of the Department of Defense to determine the feasibility of a standard data processing language that would be hardware independent. Responsibility for the study was given to the newly formed Committee of Data Systems Languages (CODASYL).

For a number of reasons, the searchers turned their backs on FACT and IBM's Commercial Translator despite their availability. They wanted a language that was not controlled by a single



manufacturer. However, FACT, FlowMatic, and the Commercial Translator system were all part of the mainstream that emerged as COBOL — COmmon Business Oriented Language.

Any tendency for a go-it-alone Tower of Babel approach was thwarted when the U.S. Government announced it would neither lease nor buy equipment unless it came equipped with COBOL — or the manufacturer could prove the machine didn't need it. Specification vagaries plagued early COBOL development. The first full version, designed by CSC for the Philco 2000, was delayed until specification questions could be ironed out.

Continuing work by CODASYL in response to operating experiences of users and software houses gradually refined specifications and today COBOL comes closer than any other language to true hardware independence.

COBOL is not a concise language — in fact, many programmers dislike the amount of writing it requires — yet, in spite of its faults, it is still the only successful commercial and universally compatible language ever developed by a cooperative of users and manufacturers.

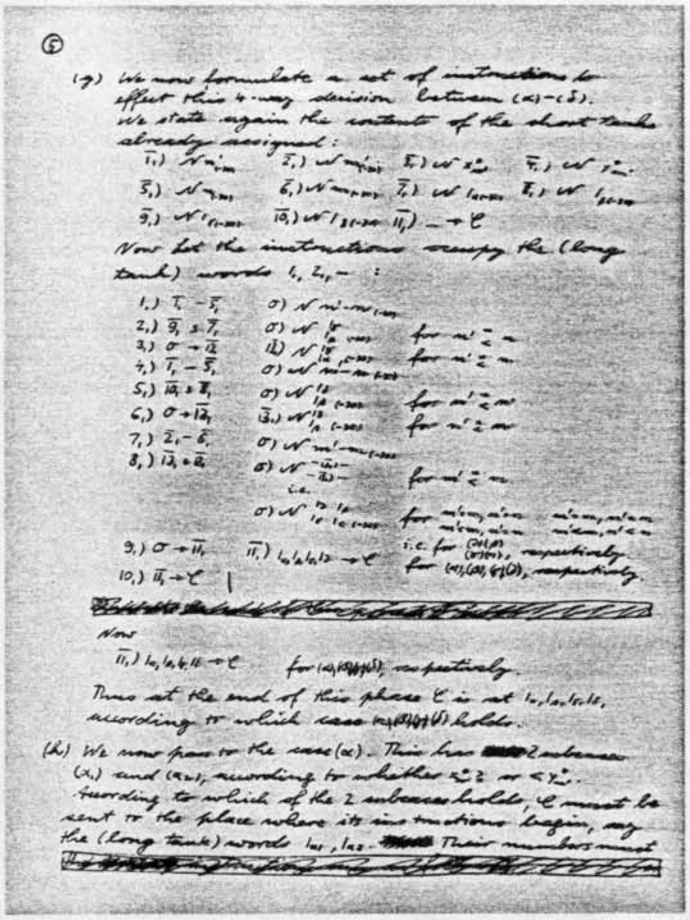
competing languages

In 1955, a group of Europeans met to plan a project similar to FORTRAN — ALGOL, or ALGOrithmic Language. Fortunately, neither effort prevented the other from making its own significant contribution to computer linguistics.

Two factors were critical to its continued development. First, it was adopted as the "common algebraic language" by an *ad hoc* committee of the Association for Computing Machinery in 1958. Their enthusiasm for the language was shared with a large group of European users. Second, the very formal and precise definition of the language not only produced a consistent syntax but also provided the impetus for serious theoretical studies of the language.

In Europe, ALGOL became the "darling of the universities," but because FORTRAN had already established itself as the primary U.S. scientific programming language, it was not widely used here.

Also, ALGOL has two serious deficiencies: a complete lack of input-output specification capabilities, and a confusing proliferation of dialects. For instance, at Burroughs a bright young programmer, Joel Erdwinn, a senior member of CSC's technical staff, wrote BALGOL, a very fast, one-pass compiler for the 220; Bendix came up with ALGOL for the G15; and the University of Michigan contributed MAD (Michigan Algorithm Decoder).



(5)

(γ) We now formulate a set of instructions to effect this 4-way decision between (α)–(δ). We state again the contents of the short tanks already assigned:

$\bar{1}_1$) [illegible] $\bar{2}_1$) [illegible] $\bar{3}_1$) [illegible] $\bar{4}_1$) [illegible]
$\bar{5}_1$) [illegible] $\bar{6}_1$) [illegible] $\bar{7}_1$) [illegible] $\bar{8}_1$) [illegible]
$\bar{9}_1$) [illegible] $\bar{10}_1$) [illegible] $\bar{11}_1$) $\ldots \to C$

Now let the instructions occupy the (long tank) words $1_1, 2_1, \ldots$:

```
1,) 1̄, − 5̄,        σ) [illegible]
2,) 9̄, ≥ 7̄,        σ) [illegible]      for [illegible]
3,) σ → 1̄2̄          1̄2̄) [illegible]     for [illegible]
4,) 1̄, − 5̄,        σ) [illegible]
5,) 1̄0̄, ≥ 8̄,       σ) [illegible]      for [illegible]
6,) σ → 1̄2̄,         1̄3̄,) [illegible]    for [illegible]
7,) 2̄, − 6̄,        σ) [illegible]
8,) 1̄3̄, ≥ 8̄,       σ) [illegible]      for [illegible]
                      i.e.
                    σ) [illegible]      for [illegible]
9,) σ → 1̄1̄,        1̄1̄,) [illegible] → C   i.e. for [illegible], respectively
10,) 1̄1̄, → C                            for [illegible], respectively
```

[illegible]

Now

$\bar{11}_1$) [illegible] $\to C$ for (α),(β),(γ),(δ), respectively.

Thus at the end of this phase C is at [illegible], according to which case (α),(β),(γ),(δ) holds.

(δ) We now pass to the case (α). This has [illegible] 2 subcases (α₁) and (α₂), according to whether [illegible] ≥ or < [illegible]. According to which of the 2 subcases holds, C must be sent to the place where its instructions begin, say the (long tank) words [illegible]. [illegible] Their numbers must

[illegible]

Facsimile of John von Neumann's original software program

In February 1959, programming work began for a large-scale command and control system for the AN/FSQ-32 military computer, using a dialect of ALGOL. The resulting algebraic compiler was known as JOVIAL, or Jules' Own Version of the International Algebraic Language, after its originator, Jules Schwartz, a principal scientist on the CSC staff.

JOVIAL was the first language to effectively handle both scientific computations and general data manipulations. Impressed, the Air Force adopted a version of JOVIAL as a standard for command and control applications. Since then, it has been adopted by the Navy and the Army for other significant programs. CSC, represented by Terry Dunbar, is a member of the Air Force *ad hoc* committee charged with resolving changes to JOVIAL.

Third-generation computers combine the capabilities required by scientists and businessmen within a single machine. To match the capability of the IBM-360 series, an Advanced Language Development Committee, under the auspices of SHARE,was formed to plan a more all-encompassing version of FORTRAN, which would still be a useful tool for engineers and scientists. Their specific goals were improved character and alphanumeric data handling, and better interaction with newer equipment and operating systems.

By the time the committee issued its report in March 1964, it became quite clear this would now become a much more powerful and complex language than FORTRAN — it was a new language, Programming Language 1. PL/1 has features more in common with COBOL and ALGOL than with FORTRAN. Its wordiness resembles COBOL, and its block structure program organization is borrowed from ALGOL.

Principally because of its ability to define and manipulate more general data structures than are available in FORTRAN, PL/1 significantly eases the job of code writing for the programmer. But, PL/1 requires a great deal of optimization to produce acceptable code, and this task is so difficult that no really "good" compiler has yet appeared.

dartmouth vs harvard

The power of third-generation machines with their on-line terminals and timesharing capabilities broadened the base of computer users. Nonprogrammers — including research scientists and business executives and their staffs — were becoming more than nodding acquaintances of the computer, and they were sorely in need of a simple language with which to voice their problems.

At Dartmouth College in 1965, a small group of undergraduate students (including Michael Busch, a senior computer scientist on the CSC professional staff) under the direction of Professors John Kemeny and Thomas Kurtz, developed BASIC — Beginner's All Purpose Symbolic Instruction Code. First implemented for the GE 225, the original BASIC contained only 15 statements which could be learned and used in a matter of hours. Since 1965, many larger variations have appeared. However, it's still easy for the nonprogrammer to master, and it's a stepping stone to other languages. Today, almost every supplier of timesharing services offers a version of BASIC.

At Harvard, Kenneth Iverson's APL — quite simply A Programming Language — was the result of a search for a clear and concise way to describe algorithms. Although originally presented in 1962, APL achieved wide acceptance only recently. It has been adopted by a number of timesharing companies and universities, and in 1970 a user's group was formed. It has proven to be far more powerful and concise than either FORTRAN or BASIC.

Operating Systems

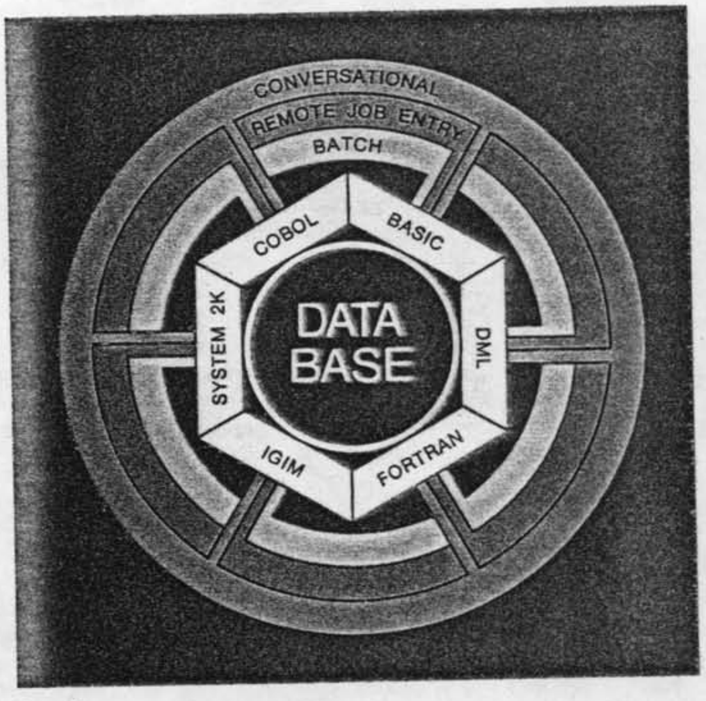
Operating systems control the overall activities of the computer in much the same way as the brain directs the actions of the human body. An operating system can be as simple or as complex as the machine it supports. The essential purposes are effective use of the computer and increased user productivity. Through a programmed repertoire of instructions, each system is capable of handling the storage, retrieval, and maintenance of data, allocating the use of hardware resources, and scheduling computing activities.

Operating systems for first-generation computers were quite elementary. When the majority of programming was done in machine language, few stored library routines were used, and most input/output operations could be handled manually. Magnetic tape records were usually sorted by transferring the data to punched cards, sorting the decks, then entering the sorted versions back onto tape. In the earliest systems, users simply included a single header card with their input deck, which assembled library routines while the program was running.

With larger and faster computers on hand, efficient operation became increasingly important. To overcome the loss of valuable processing time while operators manually loaded tapes and logged in files and programs, systems programmers at North American Aviation — with Owen Mock as principal architect — developed the first operating system for the 704. Using standard subroutines and symbolic device addressing techniques, the elementary operating system became an essential factor in "teaching" the computer to manage its own operating functions.

Mock's operating system, then, became the philosophical basis for the 709 operating system, a joint development by Mock and Donald Shell of General Electric. As a play on the names of the developers, it became known as the "MockDonald" system. With the system's EIO (Execution, Input, Output) capabilities, the pun on "Old MacDonald's Farm" happened quite naturally.

The SHARE Operating System — sos — is significant in that it was the first system to subordinate operating functions to a "supervisor" — and, according to one of its developers, it had the distinction of being one of the first major software systems to be out of control. A batch system developed by the SHARE 709 committee, it used a small monitor to determine the best order of execution, using system components as needed, until the entire load was processed. Another sos contribution was



Data base concept

macro-instruction debugging and an incremental assembler. To circumvent the relatively slow speed of the on-line card reader and printer, a multijob stream could be written off-line onto tape.

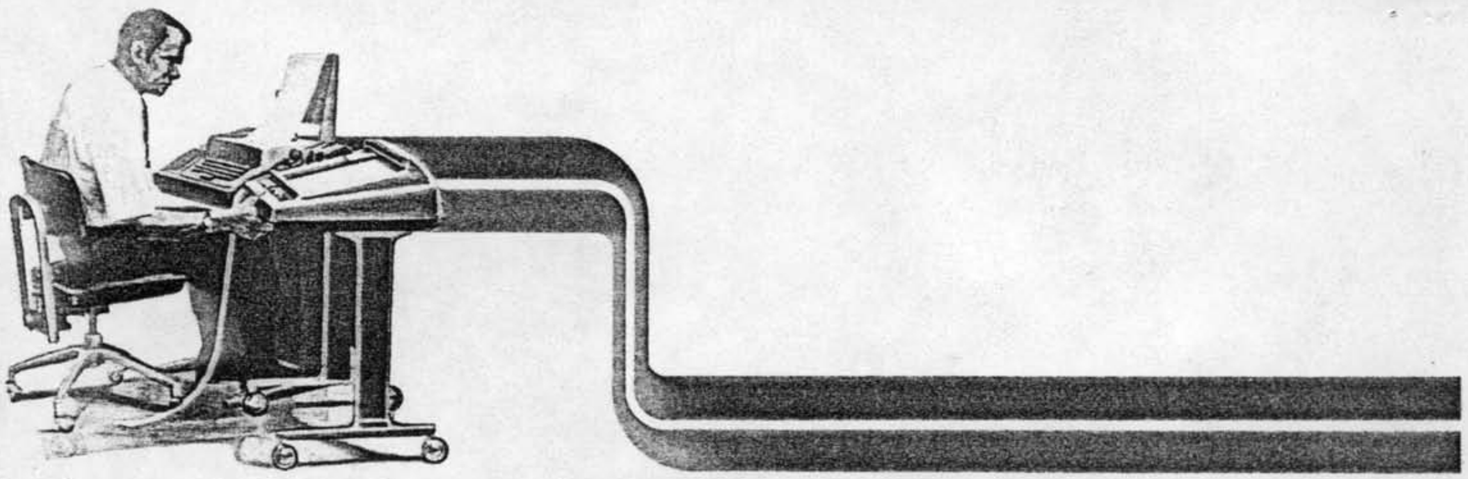
hardware influence on operating systems

Hardware has played a reciprocal role with respect to software in the development of program control. In terms of input-output development, machines up through the IBM 704 required that all high-output instructions be programmed word-for-word with the computer acting effectively as a slave of the I/O device. As a result, all programming was a linear, single-thread sequence of instructions with heavy use of subroutines.

With the arrival of the IBM 705, 709, and comparable machines, the concept of the hardware I/O channel was introduced. The computer was now able to simultaneously compute and perform I/O. Software was soon developed to exploit this in its full generality. The buffering system developed by Owen Mock and Charles Swift and used in the sos was a highly sophisticated example.

The first machines with simultaneous I/O did not have any automatic interruption of processing on completion of I/O requests, and as a result, operating systems were required to periodically poll for completion. When, with the advent of second generation machines, I/O interrupts became available, applicable software followed. For example, sos was designed in anticipation of interrupts.

Similarly, a combination of three hardware developments led to spooling: sufficient on-line storage, high speed on-line card readers and printers, and rudimentary protection. The first high-speed printers and card readers were built to operate directly to or from magnetic tape. On-line equipment was almost an order



of magnitude slower. As a result, most operating systems of this era (1955-1962) accepted magnetic tapes as program input which was written off-line from cards, and then prepared tapes to be printed. The North American/General Motors 704 System and SOS are good examples of this type of operation.

With these hardware developments it became feasible to print and read on-line. The computer still outpaced the I/O, so that data for printing usually was collected on files to be printed later by symbionts. In 1962, program protection was rudimentary, and it was considered not feasible by many to operate more than one main program simultaneously, although the main user program operated simultaneously with several symbiont processes. The Univac 1107 EXEC II system characterizes this class of systems.

Continuing the trend, software need was followed by hardware development and better hardware protection, facilitating full multiprogramming in the IBM 360 and Univac EXEC VIII systems. With communication lines and conversational terminals added, the ingredients for timesharing were assembled.

Virtual storage is one of the "new" hardware concepts affecting software designs today. From a programmer's viewpoint, virtual memory allows him to write a much larger program without concern for storage constraints. Typically, MULTICS, cooperatively developed by Massachusetts Institute of Technology, Bell Telephone Laboratories, and General Electric Company, is based on virtual storage.

the management of data...

Data management, in common with the control structure of operating systems, has been strongly influenced by hardware development. With small drums and inadequate tape drives as on the IBM 701, true data management was almost non-existent. Subsequent tape drives permitted the handling of large files. The management of reels of tape containing these files was necessarily manual. One of the earlier data management developments was tape labeling.

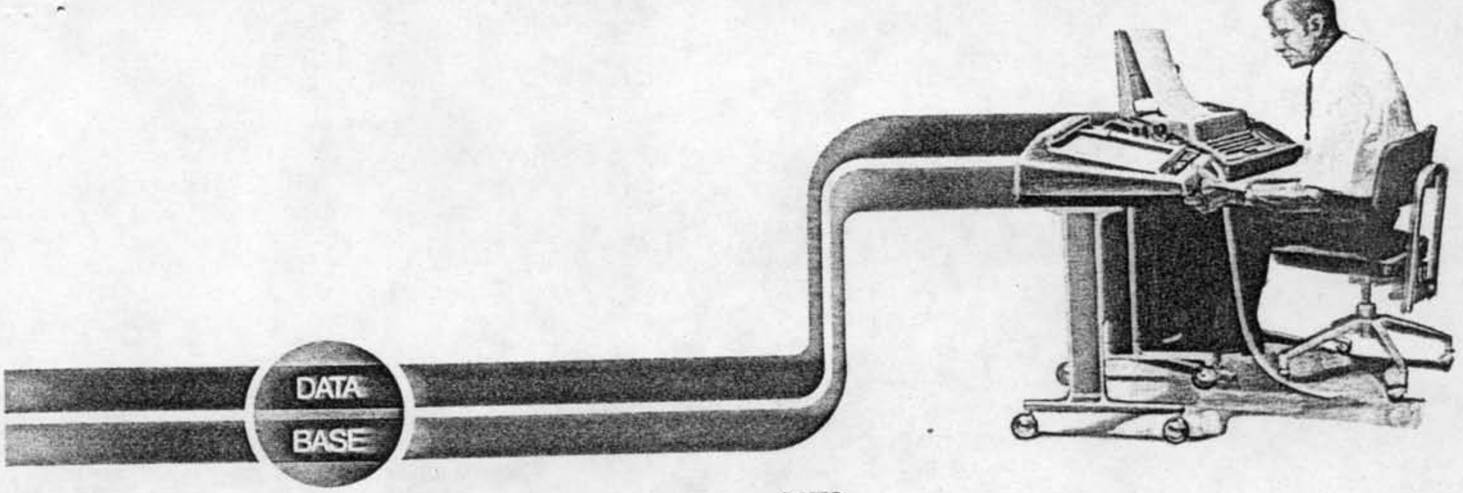
At the same time, the contents of these files began to develop structure. 9PAC on the 709 developed by the SHARE Committee, SOS in its input and output files, and FACT all operated on structured files. 9PAC and FACT files were both highly structured.

One of the problems introduced by extensive use of files was that of organizing and managing the increasing amounts of data in an efficient, easy-to-learn, consistent way. SURGE — Sorting, Updating, Report Generating — was one of the early attempts by a group of 704 users to devise a method of speeding the preparation of complete reports from brief descriptions of data content and format.

Two members of the original SURGE committee, Robert Paul, director of CSC's Software Sciences Operation, and Kerry White, a senior member of the CSC professional staff, applied the same basic concepts in developing the FACT report writer and subsequent systems.

Mass storage was required for further data management development. From 1960, with mass storage came the concept of a random access file, partially random access file (indexed sequential), and a catalog of files. It now became feasible for the computer to have permanent file memory independent of manual intervention. Finally, the concepts of file sharing and the protection concerns that ensue resulted in comprehensive data management systems, such as IBM's IMS and the specifications proposed by CODASYL's Data Base Task Group.

Timesharing systems with shared data bases have paced the development of more automatic schemes for data management. DML, a facility of the INFONET timesharing service, is typical of this new type of language processor. It incorporates file management commands with arithmetical, logical, relational, and input/output statements, and allows the user to create, retrieve, and manipulate data without concern for the physical structure of the files or the mechanics involved in processing them.



A data management system enables a user to take a single transaction or entry — without regard for any application programs — and radiate it throughout the data base. Large economies of data entry are effected by capturing the data once and triggering it to all other related records. Also, major efficiencies and reliabilities are attained on processing and retrieving data-base stored information.

...and of programs

At first glance command language appears to be a comparatively new development. In fact, however, only the rigorization and generalization in some current timesharing systems are new. Command languages were born as soon as it was necessary to separate data decks or program decks by control cards and still run them in subsequently. Command language, therefore, probably dates back at least as far as the card programmed calculator, if not farther.

Rudimentary command language existed in SOS and was further developed in the 1107 EXEC II System. The IBM 360 Operating System has a well-developed command language from a facility viewpoint if not human factors. Traditionally, batch programming language designers assumed the control cards would be prepared infrequently and paid little attention to their syntax. Conversational usage dictated that more attention be paid to ease of input of commands and a consequent new interest in command language.

One of the most interesting developments of IBM's 360/67 TSS was the macro instruction capability of its command language based on a concept of Dr. Gordon Rice of CSC. INFONET's CSTS carried this a step further, injecting much of the macro capability of a sophisticated assembler into its command language.

CSTS

The groundwork for today's large-scale, nationwide computer utility networks was laid in 1964 with the development of CSC's REMOTRAN — a batch operating processor that allowed users to communicate with a central computer from remote terminals.

This shared computer concept preceded similar manufacturer-supported systems such as the General Electric 625/35 series of computers by two years. Through successive generations the original REMOTRAN concept was enlarged and broadened into CSC's remote teleprocessing system, INFONET.

In 1967, CSC initiated a study to assess the nature of potential users' requirements for a comprehensive remote-computing service. Their findings showed that no true conversationally oriented operating system existed. Some systems offered large batch capabilities with no interactive service or vice versa; some provided an initial offering of either batch or interactive service, with patchwork-type add-on capabilities; and others were so complex that only experienced programmers could use them effectively.

In 1969, CSC started the design of a general-purpose teleprocessing system for the Univac 1108 — CSTS, or Computer Sciences Teleprocessing System*. CSTS was completed within six months of its target date, within its anticipated budget, and met all of the original study's design criteria — something of a unique landmark in the experience of developing major systems and a tribute to its project leader, Carroll Reed, and his team. The CSTS "subsystem" concept is particularly interesting, as it allows several software systems — each with its own user interface language, software library, accounting system, security, and failure recovery provisions — to coexist within the one operating system. Thus, it appears to its users as if it were several different timesharing systems at once. CSTS permits teleprocessing subscribers free choice of batch and conversational processing.

Users can select the most cost-effective mode of operation for their programs, use identical input for conversational and batch jobs, and address all files in either mode.

By allowing the free intermingling of data management languages, an environment is provided which enables data base managers to choose either DML — INFONET's Data Management Language — an elegant but easy to learn system, or the two highly advanced systems, the INFONET/TRW Generalized Information System — IGIM — and MRI's System 2000.

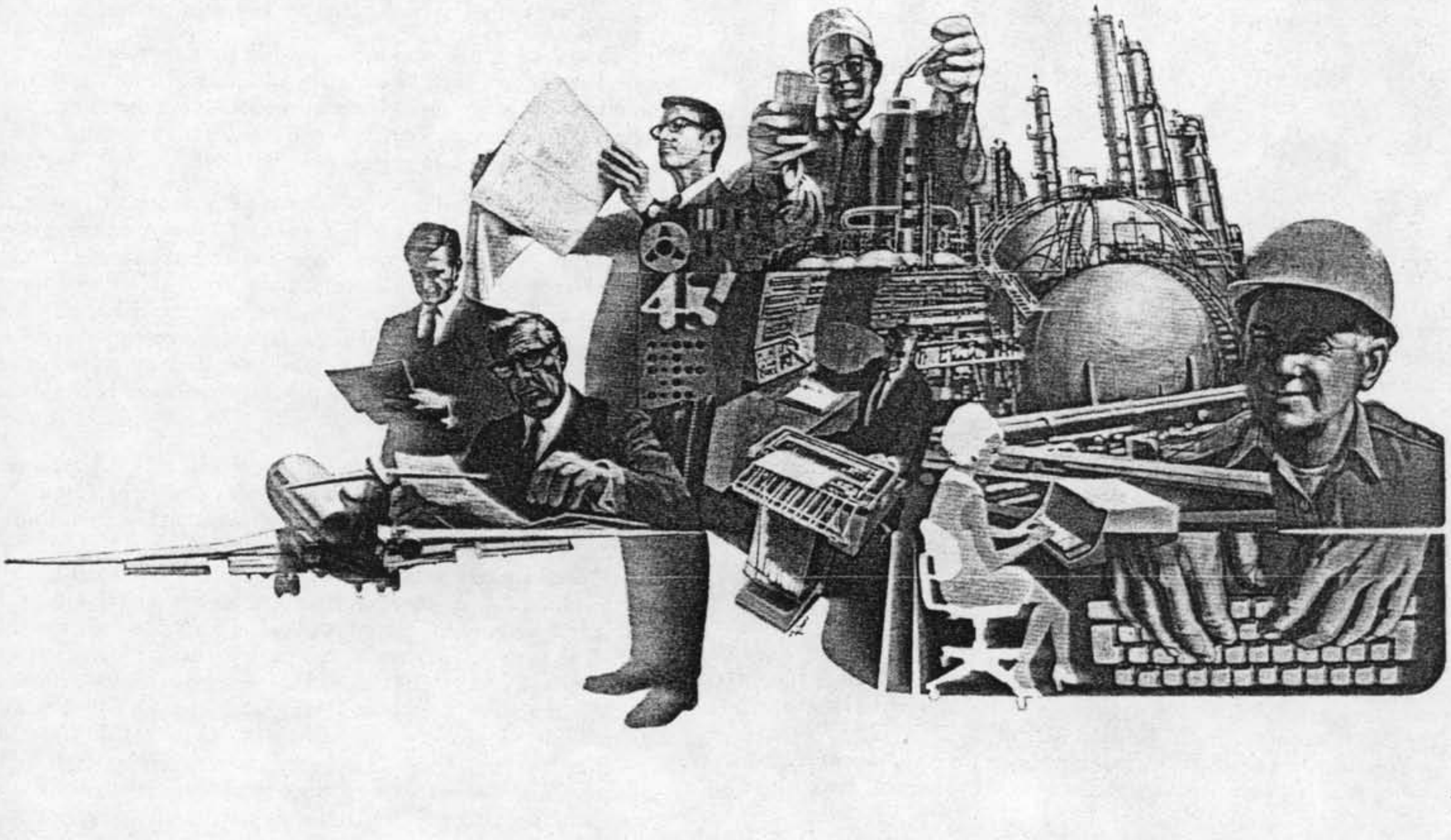
*Operating System for CSC's Information Network (INFONET) Division

To those intimate with data processing developments, there is no culmination to the evolution of software. All signs apparent at this time in the industry's growth point to continuing development of software structures that will build on the landmarks already established by CSTS, DML, IGIM, *and System 2000.*

The growth of more powerful operating systems, integrated data bases, and natural language applications is opening computer access to increasing numbers of users — skilled and unskilled. This expansion of data processing concepts now allows the computer to handle more of its own tasks with less error-prone human interface. The computer also is being applied to simulate, test, and develop its own higher level languages.

The combination of data management systems and remote teleprocessing will probably be the most significant contribution of this era to information science. First, because it makes computer access more convenient and more available, and second, because it slows the proliferation of hardware installations.

Looking ahead often entails comparison with the past. Consider the impact of the printing press on the mass transfer of information and ideas over the last four centuries. The computer in a much shorter timespan has already outperformed the printing press in quantity if not in quality. Now the alliance of communications and data processing systems has opened a whole new field of opportunities. Remote users can have unlimited access to virtually unlimited common data bases.



bibliography

Eames, Charles and Ray .... "A Computer Perspective." Harvard University Press, 1973.

Higman, Bryan .... "A Comparative Study of Programming Languages." American Elsevier Publishing, Inc., 1967.

Katzan, Harry, Jr. .... "Introduction to Programming Languages." Auerbach Publishers, Inc., 1973.

Knuth, Donald E. ..... "Fundamental Algorithms: The Art of Computer Programming." Addison-Wesley Publishing Company, 1973.

North American Aviation, Inc. .... "A Data Processing Compiler for the IBM 704." Columbus, Ohio, 1959.

Rosen, Saul .... "Programming Systems and Languages." McGraw-Hill, Inc., 1967.

Sammet, Jean E. .... "Programming Languages: History and Fundamentals." Prentice-Hall, Inc., 1969.

Wooldridge, Susan .... "Software Selection." Auerbach Publishers, Inc., 1973.

trade publication articles

Bauer, Walter F. & Rosenberg, Arthur M. .... "Software — Historical Perspectives and Current Trends." Fall Joint Computer Conference, 1972.

Frampton, Lois .... "How Does PL/1 Compare with its Forebears?" Computer Decisions, May 1970.

Knuth, Donald E. .... "The History of Sorting." Datamation, December 1972.

Lipp, Michael F. .... "The Language BASIC and its Role in Timesharing." Computers and Automation, October 1969.

McCracken, Daniel D. .... "Whither APL?" Datamation, September 15, 1970.

Rosen, Saul .... "Electronic Computers: A Historical Survey." Computer Surveys, Vol. 1, No. 1, March 1969.

Rosin, Robert F. .... "'Supervisory and Monitor Systems." Computing Surveys, Vol. 1, No. 1, March 1969.

Tropp, Henry .... "The Effervescent Years: A Retrospective" IEEE Spectrum, Computer Report VII, February 1974.

Wells, Mark B. .... "Evolution of Computer Software." Talk presented at the Los Alamos Scientific Laboratory Colloquium, January 1971.

Wilkes, Maurice V. .... "Historical Perspectives — Computer Architecture." Fall Joint Computer Conference, 1972.

acknowledgements

It goes without saying that an effort of this type is based on a great deal of painstaking research, collection of personal reminiscences, and technical verification. While a wide cross-section of CSC's systems programming staff made a valuable contribution, special thanks are due to Roy Nutt and Owen Mock for their assistance in checking historical details and participating in many informative discussions.

Robert Paul, CSC's first staff member, supplied the necessary enthusiasm, encouragement, advice, and direction as the publication gradually took shape. And Business Promotion Group editors, Morton Bernheim, Susan Thorne, and John Boland, deserve acknowledgement for preparing and coordinating masses of collected material, cross-checking, and editing the contributions of many technical contributors and reviewers.

Last — but by no means least — thanks are due to Terry Vincelette, who patiently typed version after version under threats that she would not be permitted to produce her first (pending) offspring until the document was in print!