

PLANNING A COMPUTER SYSTEM

PROJECT STRETCH

PLANNING A

CONTRIBUTORS

- Richard S. Ballance
Robert W. Bemer
Gerrit A. Blaauw
Erich Bloch
Frederick P. Brooks, Jr.
Werner Buchholz
Sullivan G. Campbell
John Cocks
Edgar F. Codd
Paul S. Herwitz
Harwood G. Kolsky
Edward S. Lowry
Elizabeth McDonough
James H. Pomerene
Casper A. Scalzi

3160
Cop. 3

COMPUTER SYSTEM

PROJECT STRETCH

Edited by
WERNER BUCHHOLZ

SYSTEMS CONSULTANT
CORPORATE STAFF, RESEARCH AND ENGINEERING
INTERNATIONAL BUSINESS MACHINES CORPORATION



New York Toronto London 1962

McGRAW-HILL BOOK COMPANY, INC.

COMPUTER SYSTEM
PLANNING A COMPUTER SYSTEM

Copyright © 1962 by the McGraw-Hill Book Company, Inc. Printed in the United States of America. All rights reserved. This book, or parts thereof, may not be reproduced in any form without permission of the publishers. *Library of Congress Catalog Card Number 61-10466*

THE MAPLE PRESS COMPANY, YORK, PA.

08720

the conversion time for input and output data intended for use in extensive mathematical computation. Decimal arithmetic is also included in the instruction repertoire, in order to permit simple arithmetical operations to be performed directly on data in binary-coded decimal form.

Such a combination of binary and decimal arithmetic in a single computer provides a high-performance tool for many diverse applications. It may be noted that a different conclusion might be reached for a computer with a restricted range of functions or with performance goals limited in the interest of economy; the difference between binary and decimal operation might well be considered too small to justify incorporating both. This conclusion does appear valid for high-performance computers, regardless of whether they are aimed primarily at scientific computing, business data processing, or real-time control. To recommend binary addressing for a computer intended for business data processing is admittedly a departure from earlier practice, but the need for handling and storing large quantities of nonnumerical data makes the features of binary addressing particularly attractive. In the past, the real obstacle to binary computers in business applications has been the difficulty of handling inherently decimal data. Binary addressing and decimal data arithmetic, therefore, make a powerful combination.

Chapter 6

CHARACTER SET

by R. W. Bemer and W. Buchholz

6.1. Introduction

Among the input and output devices of a computer system, one can distinguish between those having built-in codes and those largely insensitive to code. Thus typewriters and printers necessarily have a fixed code that represents printable symbols to be read by the human eye; a code must be chosen for such a device in some more or less arbitrary fashion, and the device must make the transformation between code and symbol. Data storage and transmission devices, on the other hand, such as magnetic tape units and telephone transmission terminals, merely repeat the coded data given to them without interpretation, except that some code combinations may possibly be used to control the transmission process. (Strictly speaking, storage and transmission devices do generally limit the code structure in some respect, such as maximum byte size, so that code sensitivity is a matter of degree.)

For the inherently code-sensitive devices to be attached to a new computer system, an obvious choice of character set and code would have been one of the many sets already established. When the existing sets were reviewed, however, none were found to have enough of the system characteristics considered desirable. In fact, it became clear that about the only virtue of choosing an already established set is that the set exists. Accordingly, it was decided, instead, to devise a new character set expressly for use throughout a modern computer system, from input to output. The chief characteristic of this set is its extension to many more different characters than have been available in earlier sets. The extended set designed for the 7030 (Fig. 6.1) contains codes for 120 different characters, but there is room for later expansion to up to 256 characters including control characters. In addition, useful subsets have been defined, which contain some but not all of these 120 characters and which use the same codes for the selected characters without translation.

It should be noted that the 7030 computer is relatively insensitive to the specific choice of code, and any number of codes could be successfully used in the system. For any particular application a specialized character code might be found superior. In practice, however, a large computer

Bits 4-5-6-7	Bits 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	Blank	[£	c	k	s	0	8
0001	±	▷	+	C	K	S	o	8
0010	→]	\$	d	l	t	1	9
0011	≠	°	=	D	L	T	1	9
0100	^	+	*	e	m	u	2	.
0101	{	≡	(E	M	U	2	:
0110	↑	¬	/	f	n	v	3	-
0111	}	√)	F	N	V	3	?
1000	v	%	,	g	o	w	4	
1001	∇	\	;	G	O	W	4	
1010	↓	◇	'	h	p	x	5	
1011			"	H	P	X	5	
1100	>	#	a	i	q	y	6	
1101	≡	!	A	I	Q	Y	6	
1110	<	@	b	j	r	z	7	
1111	≡	~	B	J	R	Z	7	

FIG. 6.1. 120-character set.

installation must deal with a mixture of widely different applications, and the designers have to choose a single character set as a compromise among conflicting requirements.

The purpose of this chapter is to list major requirements of a character set and code, and to point out how these requirements may or may not be met by the specific set to be described.

6.2. Size of Set

Present IBM 48-character sets consist of

1. 10 decimal digits
2. 26 capital letters
3. 11 special characters
4. 1 blank

Other manufacturers have employed character sets of similar or somewhat larger size.

Because a single set of eleven special characters is not sufficient, there exist several choices of special characters as "standard options."

Since this 48-character set is often represented by a 6-bit code, it is natural to try to extend it to 63 characters and a blank, so as to exploit the full capacity of a 6-bit code.¹ Although the extra sixteen characters would indeed be very useful, this step was thought not to be far-reaching enough to justify development of the new equipment that it would require.

As a minimum, a new set should include also:

5. 26 lower-case letters
6. The more important punctuation symbols found on all office typewriters
7. Enough mathematical and logical symbols to satisfy the needs of such programming languages as ALGOL^{2,3}

There is, of course, no definite upper limit on the number of characters. One could go to the Greek alphabet, various type fonts and sizes, etc., and reach numbers well into the thousands. As set size increases, however, cost and complexity of equipment go up and speed of printing goes down. The actual choice of 120 characters was a matter of judgment; it was decided that this increment over existing sets would be sufficiently large to justify a departure from present codes and would not include many characters of only marginal value.

6.3. Subsets

Two subsets of 89 and 49 characters were chosen for specific purposes. The 89-character set (Fig. 6.2) is aimed at typewriters, which, with 44

¹ H. S. Bright, Letter to the Editor, *Communs. ACM*, vol. 2, no. 5, pp. 6-9, May, 1959 (a 64-character alphabet proposal).

² A. J. Perlis and K. Samelson, Preliminary Report: International Algebraic Language, *Communs. ACM*, vol. 1, no. 12, December, 1958.

³ Peter Naur (editor), Report on the Algorithmic Language ALGOL 60, *Communs. ACM*, vol. 3, no. 5, May, 1960.

character keys, a case shift, and a space bar, can readily handle 89 characters. This subset was considered important because input-output typewriters can already print 89 characters without modification, and 44-key keyboards are familiar to many people.

The 49-character subset (Fig. 6.3) is the conventional set of "commercial" characters in a code compatible with the full set.¹ This subset is aimed at the *chain printer* mechanism used with the 7030, which can readily print character sets of different sizes but prints the larger sets at a reduced speed. The 49-character subset permits high-volume printing at high speed in a compatible code on jobs (such as bill printing) where the extra characters of the full set may not be needed. It should be noted that the 49-character set is not entirely a subset of the 89-character set.

Other subsets are easily derived and may prove useful. For example, for purely numerical work, one may wish to construct a 13-character set consisting of the ten digits and the symbols . (point) and - (minus), together with a special blank.

6.4. Expansion of Set

Future expansion to a set larger than 120 can take place in two ways.

One is to assign additional characters to presently unassigned codes; allowance should then be made for certain control codes which will be needed for communication and other devices and which are intended to occupy the high end of the code sequence.

The second way is to define a shift character for "escape" to another character set.² Thus, whenever the shift character is encountered, the next character (or group of characters) identifies a new character set, and subsequent codes are interpreted as belonging to that set. Another shift character in that set can be used to shift to a third set, which may again be the first set or a different set. Such additional sets would be defined only if and when there arose applications requiring them.

6.5. Code

In choosing a code structure, many alternatives were considered. These varied in the basic number of bits used (i.e., the byte size) and in the number of such bytes that might be used to represent a single (print-

¹ Note that this is one character larger than the previously referred-to 48-character set. The additional special character was introduced in 1959 on the printer of the IBM 1401 system; but its use has not become firmly established, partly because it has no counterpart on the keypunch. Thus the 48- and 49-character sets are, in effect, the same set.

² R. W. Bemer, A Proposal for Character Code Compatibility, *Communs. ACM*, vol. 3, no. 2, February, 1960.

Bits 4-5-6-7	Bits 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	Blank		ε	c	k	s	0	8
0001			+	C	K	S	o	8
0010			\$	d	l	t	1	9
0011			=	D	L	T	1	9
0100			*	e	m	u	2	.
0101			(E	M	U	2	:
0110			/	f	n	v	3	-
0111)	F	N	V	3	?
1000			,	g	o	w	4	
1001			;	G	O	W	4	
1010			'	h	p	x	5	
1011			"	H	P	X	5	
1100			a	i	q	y	6	
1101			A	I	Q	Y	6	
1110			b	j	r	z	7	
1111			B	J	R	Z	7	

FIG. 6.2. 89-character set.

able) character. Among the alternatives were the following:

Single 6-bit byte with shift codes interspersed

Double 6-bit byte = single 12-bit byte¹

Single 8-bit byte

Single 12-bit byte for "standard" characters (punched-card code) and two 12-bit bytes for other characters

Some of these codes represented attempts to retain partial compatibility with earlier codes so as to take advantage of existing equipment.

¹ R. W. Bemer, A Proposal for a Generalized Card Code for 256 Characters, *Communications. ACM*, vol. 2, no. 9, September, 1959.

0123|4567

Bits 4-5-6-7	Bits 0-1-2-3							
	0000	0001	0010	0011	0100	0101	0110	0111
0000	Blank		£				0	8
0001				C	K	S		
0010			\$				1	9
0011				D	L	T		
0100			*				2	.
0101				E	M	U		
0110			/				3	-
0111				F	N	V		
1000		%	,				4	
1001				G	O	W		
1010		◇	'				5	
1011				H	P	X		
1100		#					6	
1101			A	I	Q	Y		
1110		@					7	
1111			B	J	R	Z		

FIG. 6.3. 49-character set.

These attempts were abandoned, in spite of some rather ingenious proposals, because the advantages of partial compatibility were not enough to offset the disadvantages.

The 8-bit byte was chosen for the following reasons:

1. Its full capacity of 256 characters was considered to be sufficient for the great majority of applications.
2. Within the limits of this capacity, a single character is represented by a single byte, so that the length of any particular record is not dependent on the coincidence of characters in that record.
3. 8-bit bytes are reasonably economical of storage space.

4. For purely numerical work, a decimal digit can be represented by only 4 bits, and two such 4-bit bytes can be packed in an 8-bit byte. Although such packing of numerical data is not essential, it is a common practice in order to increase speed and storage efficiency. Strictly speaking, 4-bit bytes belong to a different code, but the simplicity of the 4-and-8-bit scheme, as compared with a combination 4-and-6-bit scheme, for example, leads to simpler machine design and cleaner addressing logic.

5. Byte sizes of 4 and 8 bits, being powers of 2, permit the computer designer to take advantage of powerful features of binary addressing and indexing to the bit level (see Chaps. 4 and 5).

The eight bits of the code are here numbered for identification from left to right as 0 (high-order bit) to 7 (low-order bit). "Bit 0" may be abbreviated to B_0 , "bit 1" to B_1 , etc.

6.6. Parity Bit

For transmitting data, a ninth bit is attached to each byte for parity checking, and it is chosen so as to provide an odd number of 1 bits. Assuming a 1 bit to correspond to the presence of a signal and assuming also an independent source of timing signals, *odd parity* permits all 256 combinations of 8 bits to be transmitted and to be positively distinguished from the absence of information. The parity bit is identified here as "bit P " or B_P .

The purpose of defining a parity bit in conjunction with a character set is to establish a standard for communicating *between* devices and media using this set. It is not intended to exclude the possibilities of error correction or other checking techniques *within* a given device or on a given medium when appropriate.

6.7. Sequence

High-equal-low comparisons are an important aspect of data processing. Thus, in addition to defining a standard code for each character, one must also define a standard comparing (*collating*) sequence. Obviously, the decimal digits must be sequenced from 0 to 9 in ascending order, and the alphabet from A to Z. Rather more arbitrary is the relationship between groups of characters, but the most prevalent convention for the 48 IBM "commercial" characters is, in order:

- (Low) Blank
- Special characters . \square & \$ * - / , % # @
- Alphabetic characters A to Z
- (High) Decimal digits 0 to 9

Fundamentally, the comparing sequence of characters should conform to the natural sequence of the binary integers formed by the bits of that

code. Thus 0000 0100 should follow 0000 0011. Few existing codes have this property, and it is then necessary, in effect, to translate to a special internal code during alphanumeric comparisons. This takes extra equipment, extra time, or both. An important objective of the new character set was to obtain directly from the code, without translation, a usable comparing sequence.

A second objective was to preserve the existing convention for the above 48 characters within the new code. This objective has not been achieved because of conflicts with other objectives.

The 7030 set provides the following comparing sequence without any translation:

- (Low) Blank
- Special characters (see chart)
- Alphabetic characters a A b B c C to z Z
- Numerical digits 0 1 to 9
- Special characters . : - ?
- (High) Unassigned character codes

Note that the lower- and upper-case letters occur in pairs in adjacent positions, following the convention established for directories of names. (There appeared to be no real precedent for the relative position within the pair. The case shift is generally ignored in the sequence of names in telephone directories, even when the same name is spelled with either upper- or lower-case letters. This convention is not usable in general, since each character code must be considered unique.)

The difference between this comparing sequence and the earlier convention lies only in the special characters. Two of the previously available characters had to be placed at the high end, and the remaining special characters do not fall in quite the same sequence with respect to one another. It was felt that the new sequence would be quite usable and that it would be necessary only rarely to re-sort a file in the transition to the 7030 code. It is always possible to translate codes to obtain any other sequence, as one must do with most existing codes.

6.8. Blank

The code 0000 0000 is a natural assignment for the *blank* (i.e., the nonprint symbol that represents an empty character space). Not only should the *blank* compare lower than any printable character, but also absence of bits (other than the parity bit) corresponds to absence of mechanical movement in a print mechanism.

Blank differs, however, from a *null* character, such as the all-ones code found on paper tape. *Blank* exists as a definite character occupying a definite position on a printed line, in a record, or in a field to be compared.

A *null* may be used to delete an erroneous character, and it would be completely dropped from a record at the earliest opportunity. *Null*, therefore, occupies no definite position in a comparing sequence. A *null* has not been defined here, but it could be placed when needed among the control characters.

Considering numerical work only, it would be aesthetically pleasing to assign the all-zeros code to the digit zero, that is, to use 0000 as the common zone bits of the numeric digits (see below). In alphanumeric work, however, the comparing sequence for *blank* should take preference in the assignment of codes.

6.9. Decimal Digits

The most compact coding for decimal digits is a 4-bit code, and the natural choices for encoding 0 to 9 are the binary integers 0000 to 1001. As mentioned before, two such digits can be packed into an 8-bit byte; for example, the digits 28 in packed form could appear as

0010 1000

If decimal digits are to be represented unambiguously in conjunction with other characters, they must have a unique 8-bit representation. The obvious choice is to spread pairs of 4-bit bytes into separate 8-bit bytes and to insert a 4-bit prefix, or *zone*. For example, the digits 28 might be encoded as

zzzz 0010 zzzz 1000

where the actual value of each zone bit *z* is immaterial so long as the prefix is the same for all digits.

This requirement conflicted with requirements for the comparing sequence and for the case shift. As a result, the 4-bit byte is offset by 1 bit, and the actual code for 28 is

0110 0100 0111 0000

This compromise retains the binary integer codes 0000 to 1001 in adjacent bit positions, but not in either of the two positions where they appear in the packed format.

The upper-case counterparts of the normal decimal digits are assigned to italicized decimal subscripts.

6.10. Typewriter Keyboard

The most commonly found devices for key-recording input to a computer system are the IBM 24 and 26 keypunches, but their keyboards are not designed for keying both upper- and lower-case alphabetic characters. The shifted positions of some of the alphabetic characters are used to punch numerical digits. For key-recording character sets with

much more than the basic 48 characters, it is necessary to adopt a keyboard convention different from that of the keypunch. The 89-character subset was established to bring the most important characters of the full set within the scope of the common typewriter, thus taking advantage of the widespread familiarity with the typewriter keyboard and capitalizing on existing touch-typing skills as much as possible.

The common typewriter keyboard consists of up to 44 keys and a separate case-shift key. To preserve this relationship in the code, the 44 keys are represented by 6 bits of the code (B_1 to B_6) and the case shift by a separate bit (B_7). The case shift was assigned to the lowest-order bit, so as to give the desired sequence between lower- and upper-case letters.

For ease of typing, the most commonly used characters should appear in the lower shift ($B_7 = 0$). This includes the decimal digits and, when both upper- and lower-case letters are used in ordinary text, the lower-case letters. (This convention differs from the convention for single-case typewriters presently used in many data-processing systems; when no lower-case letters are available, the digits are naturally placed in the same shift as the upper-case letters.) It is recognized that the typewriter keyboard is not the most efficient alphanumeric keyboard possible, but it would be unrealistic to expect a change in the foreseeable future. For purely numerical data, it is always possible to use a 10-key keyboard either instead of the typewriter keyboard or in addition to it.

It was not practical to retain the upper- and lower-case relationships of punctuation and other special characters commonly found on typewriter keyboards. There is no single convention anyway, and typists are already accustomed to finding differences in this area.

6.11. Adjacency

The 52 characters of the upper- and lower-case alphabets occupy 52 consecutive code positions without gaps. For the reasons given above, it was necessary to spread the ten decimal digits into every other one of twenty adjacent code positions, but the remaining ten positions are filled with logically related decimal subscripts. The alphabet and digit blocks are also contiguous. Empty positions for additional data and control characters are all consolidated at the high end of the code chart.

This grouping of related characters into solid blocks of codes, without empty slots that would sooner or later be filled with miscellaneous characters, assists greatly in the analysis and classification of data for editing purposes. Orderly expansion is provided for in advance.

6.12. Uniqueness

A basic principle underlying the choice of this set is to have only one code for each character and only one character for each code.

Much of the lack of standardization in existing character sets arises from the need for more characters than there are code positions available in the keying and printing equipment. Thus, in the existing 6-bit IBM character codes, the code 001100 may stand for any one of the three characters @ (at), - (minus), and ' (apostrophe). The 7030 set was required to contain all these characters with a unique code for each.

The opposite problem exists too. Thus, in one of the existing 6-bit codes, - may be represented by either 100000 or 001100. Such an embarrassment of riches presents a logical problem when the two codes have in fact the same meaning and can be used interchangeably. No amount of comparing and sorting will bring like items together until one code is replaced by the other everywhere.

In going to a reasonably large set, it was necessary to resist a strong temptation to duplicate some characters in different code positions so as to provide equal facilities in various subsets. Instead, every character has been chosen so as to be typographically distinguishable if it stands by itself without context. Thus, for programming purposes, it is possible to represent any code to which a character has been assigned by its unique graphic symbol, even when the bit grouping does not have the ordinary meaning of that character (e.g., in operation codes).

In many instances, however, it is possible to find a substitute character close enough to a desired character to represent it in a more restricted subset or for other purposes. For example, = (equals) may stand for ← (is replaced by) in an 89-character subset. Or again, if a hyphen is desired that compares lower than the alphabet, the symbol ~ (a modified tilde) is preferred to the more conventional - (minus).

A long-standing source of confusion has been the distinction between upper-case "oh" (O) and zero (0). Some groups have solved this problem by writing zero as Ø. Unfortunately, other groups have chosen to write "oh" as Ø. Neither solution is typographically attractive. Instead, it is proposed to modify the upper-case "oh" by a center dot (leaving the zero without the dot) and to write and print "oh" as Θ whenever a distinction is desired.

Various typographic devices are used to distinguish letters (I, l, V, etc.) from other characters [| (stroke), 1 (one), ∨ (or), etc.]. It is suggested that the italicized subscripts be underlined when handwritten by themselves, for example, $\underline{\dot{z}}$.

6.13. Signs

The principle of uniqueness implies a separate 8-bit byte to represent a plus or a minus sign. Keying and printing equipment also require separate sign characters. This practice is, of course, rather expensive in storage space, but it was considered superior to the ambiguity of present

6-bit codes where otherwise "unused" zone bits in numerical fields are used to encode signs. If the objective is to save space, one may as well abandon the alphanumeric code quite frankly and switch to a 4-bit decimal coding with a 4-bit sign digit, or go to the even more compact binary radix.

6.14. Tape-recording Convention

As has been remarked before, data-recording media such as magnetic tape and punched cards are not inherently code-sensitive. It is obviously necessary, though, to adopt a fixed convention for recording a code on a given medium if that medium is to be used for communication between different systems.

Magnetic tape with eight, or a multiple of eight, information tracks permits a direct assignment of the 8 bits in the 7030 code to specific tracks. Magnetic tape with six information tracks requires some form of *byte conversion* to adapt the 8-bit code to the 6-bit tape format. The convention chosen is to distribute three successive 8-bit bytes over four successive 6-bit bytes on tape. This convention uses the tape at full efficiency, leaving no gaps except possibly in the last 6-bit byte, which may contain 2 or 4 nonsignificant 0 bits, depending on the length of the record.

Thus successive 8-bit bytes, each with bits B_0 to B_7 , are recorded as shown in Table 6.1.

TABLE 6.1. CONVENTION FOR RECORDING 8-BIT CODE ON 6-TRACK TAPE

Track	Bits
0	B_0 B_6 B_4 B_2 B_0
1	B_1 B_7 B_5 B_3 B_1
2	B_2 B_0 B_6 B_4 B_2
3	B_3 B_1 B_7 B_5 B_3 etc.
4	B_4 B_2 B_0 B_6 B_4
5	B_5 B_3 B_1 B_7 B_5

The parity bit is not shown. The parity bits for the 6-bit tape format are, of course, different from those of the 8-bit code; so parity conversion must be provided also.

6.15. Card-punching Convention

Since 80-column punched cards are a common input medium, a card-punching convention for the 120 characters is likewise desirable. After the possibility of a separate card code for the 120 characters was considered—a code having the conventional IBM card code as a subset¹—

¹ *Ibid.*

it was concluded that it would be better to punch the 8-bit code directly on the card. This does not preclude also punching the conventional code (limited to 48 characters) on part of the card for use with conventional equipment. Code translation is then needed only whenever the conventional card code is used; otherwise translation would be required for every column if advantage is to be taken of the new code in the rest of the system.

The punching convention is given in Table 6.2.

In addition, both hole 12 and hole 11 are to be punched in column 1 of every card containing the 7030 code, besides a regular 7030 character, so as to distinguish a 7030 card from cards punched with the conventional code. Eight-bit punching always starts in column 1 and extends as far as desired; a control code END (0 1111 1110) has been defined to terminate the 8-bit code area. Conventional card-code punching should

TABLE 6.2. CONVENTION FOR PUNCHING 8-BIT CODE ON CARDS

<i>Card row</i>	<i>Bit</i>
12	—
11	—
0	—
1	B_P
2	B_0
3	B_1
4	B_2
5	B_3
6	B_4
7	B_5
8	B_6
9	B_7

be confined to the right end of those cards identified with 12-11 punching in column 1.

Since the parity bit is also punched, the 7030 area of a card contains a checkable code. Note that "blank" columns in this area still have a hole in the B_P row. If only part of the card is to be punched, however, it is possible to leave the remaining columns on the right unpunched.

6.16. List of 7030 Character Set

A list of the 7030 character-set codes and graphic symbols is shown for reference in Fig. 6.4, which includes the names of the characters.

Code		Character Name	Code	
P 0123 4567			P 0123 4567	
1 0000 0000		Blank (Space)	0 0010 0000	& Ampersand
0 0000 0001	±	Plus or minus	1 0010 0001	+ Plus sign
0 0000 0010	-	Right arrow (Replaces)	1 0010 0010	\$ Dollar sign
1 0000 0011	≠	Not equal	0 0010 0011	= Equals
0 0000 0100	∧	And	1 0010 0100	* Asterisk (Multiply)
1 0000 0101	{	Left brace	0 0010 0101	(Left parenthesis
1 0000 0110	↑	Up arrow (Start super- script)	0 0010 0110	/ Right slant (Divide)
0 0000 0111	}	Right brace	1 0010 0111) Right paren- thesis
0 0000 1000	∨	Or (inclusive)	1 0010 1000	,
1 0000 1001	∩	Exclusive Or	0 0010 1001	;
1 0000 1010	↓	Down arrow (End super- script)	0 0010 1010	' Apostrophe (Single quote)
0 0000 1011		Double lines	1 0010 1011	" Ditto (Double quote)
0 0000 1100	>	Greater than	0 0010 1100	a
0 0000 1101	≥	Greater than or equal	1 0010 1101	A
0 0000 1110	<	Less than	1 0010 1110	b
1 0000 1111	≤	Less than or equal	0 0010 1111	B
0 0001 0000	[Left bracket	1 0011 0000	c
1 0001 0001	⊃	Implies	0 0011 0001	C
1 0001 0010]	Right bracket	0 0011 0010	d
0 0001 0011	°	Degree	1 0011 0011	D
1 0001 0100	←	Left arrow (Is replaced by)	0 0011 0100	e
0 0001 0101	≡	Identical	1 0011 0101	E
0 0001 0110	¬	Not	1 0011 0110	f
1 0001 0111	√	Square root (Check mark)	0 0011 0111	F
1 0001 1000	%	Percent sign	0 0011 1000	g
0 0001 1001	\	Left slant (Re- verse divide)	1 0011 1001	G
0 0001 1010	◇	Lozenge (Dia- mond)(Note)	1 0011 1010	h
1 0001 1011		Absolute value (Vertical line)	0 0011 1011	H
0 0001 1100	#	Number sign	1 0011 1100	i
1 0001 1101	!	Exclamation point (Fac- torial)	0 0011 1101	I
1 0001 1110	@	At sign	0 0011 1110	J
0 0001 1111	~	Tilde (Hyphen)	1 0011 1111	J

Note: The character □ has also been used.

FIG. 6.4. List of 7030 codes and characters. (Continued on next page.)

Code		Character Name	Code		Character Name
P	0123 4567		P	0123 4567	
0	0100 0000	k	1	0110 0000	0 Zero
1	0100 0001	K	0	0110 0001	0 Subscript zero
1	0100 0010	l	0	0110 0010	1 One
0	0100 0011	L	1	0110 0011	1 Subscript one
1	0100 0100	m	0	0110 0100	2 Two
0	0100 0101	M	1	0110 0101	2 Subscript two
0	0100 0110	n	1	0110 0110	3 Three
1	0100 0111	N	0	0110 0111	3 Subscript three
1	0100 1000	o	0	0110 1000	4 Four
0	0100 1001	0	1	0110 1001	4 Subscript four
0	0100 1010	p	1	0110 1010	5 Five
1	0100 1011	P	0	0110 1011	5 Subscript five
0	0100 1100	q	1	0110 1100	6 Six
1	0100 1101	Q	0	0110 1101	6 Subscript six
1	0100 1110	r	0	0110 1110	7 Seven
0	0100 1111	R	1	0110 1111	7 Subscript seven
1	0101 0000	s	0	0111 0000	8 Eight
0	0101 0001	S	1	0111 0001	8 Subscript ei
0	0101 0010	t	1	0111 0010	9 Nine
1	0101 0011	T	0	0111 0011	9 Subscript nine
0	0101 0100	u	1	0111 0100	. Period (point)
1	0101 0101	U	0	0111 0101	: Colon
1	0101 0110	v	0	0111 0110	- Minus sign
0	0101 0111	V	1	0111 0111	? Question mark
0	0101 1000	w			
1	0101 1001	W			
1	0101 1010	x			
0	0101 1011	X			
1	0101 1100	y			
0	0101 1101	Y			
0	0101 1110	z			
1	0101 1111	Z			

FIG. 6.4 (Continued)

THE AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE

by R. W. BEMER, UNIVAC Division, Sperry Rand Corp.,
New York, N.Y.

It is very doubtful that Herman Hollerith ever considered, in 1905, that he would have to talk to Jean Baudot. After all, the man was dead. But this is exactly what is happening today in the inevitable marriage of computers and communications systems. The punched cards that Hollerith created must communicate with the punched paper tape of Baudot. The problem is that there is absolutely no logical similarity or relationship between the codes which represent the various letters, digits and other characters.

Hollerith designed his code for a mechanical counting reader. When cards became input to computers, as well as mechanical devices, a code correspondence had to be applied. In forming the IBM binary coded decimal (BCD) code, the 0 to 9 rows on the card were equated to 0000 through 1001, and thus the binary value corresponds to the decimal row value. Two more bits precede these to represent the four "zones" (12, 11, 0 and blank) by 00, 01, 10, and 11, although not respectively and indeed this varies among IBM equipment. Other manufacturers made different assignments in various attempts for internal economies. Assignments even vary among individual customers. Thus although most IBM users have the 12 punch as a plus sign and the 11 punch as a minus sign there are many others to whom the reverse is true.

There is a binary code inherent in the punched paper tape of Baudot, but this depends upon which tracks are made to correspond to which binary positions. Sorry to say, this choice has been made in several ways. Even so, Baudot did not make his assignment on a sequential basis for the digits or letters of the alphabet. Due to the technology of the time, it was done on the basis that the most frequently used characters would be represented by the fewest punched holes, to save wear and tear on the punch dies! To illustrate:

Letter	blank	E	T	A	O	I	N	S	H	R	D	L	U
No. of Punches	1	1	1	2	2	2	2	2	2	2	2	2	3

This should prove that there was never an actual person

part one:
review and preview

by that name! Apparently not much was known about digit frequency in those days, for they were assigned:

Digit	0	1	2	3	4	5	6	7	8	9
No. of Punches	3	4	3	1	2	1	3	3	2	2

Such technological conditions are largely removed now, and logical considerations assume commanding importance.

topsy in the information processing field

About four or five years ago many people awoke individually to the fact that we are in an almost impossible jumble in the coding of information. Consider the way it grew from the IBM standpoint, based upon the punched card. First came the digits 0 to 9, with & (or +) and -. So far there is only one problem, the duality of & and + as represented by the 12-punch. Now add zones for the letters. The digits 0 to 9 with a 12-punch mean A to I, with an 11-punch they mean J to R, and with a 0-punch



Mr. Bemer is an alternate member of X3.2, the X3.5 Glossary committee, and AFIP member of the IFIP TC-1 Terminology committee. Before joining Univac, where he is director of systems programming, he was with The RAND Corp. and IBM, and started the computing installation at Lockheed Missiles.

they mean / and S to Z. Simple. But now what? Without using the other combinations of two punches (e.g. 3-6) we move directly to combinations of three punches by adding 8's. This gives such characters as . , * \$ @ & and ◊. So far this can be lived with.

Now design the reader on the IBM 702 so that all illegal punch combinations are rejected; that is, only 48 out of the 4096 (2¹²) possible combinations are legal. Now find out that more codes are needed for tape control in a computer, and so far only 48 out of a possible 64 codes available in a 6-bit character have been used. We try to see what happens for all 4096 combinations, and are surprised to find that the engineer goofed a little — nine supposedly illegal combinations slip by! So 0-2-8 is a record mark and 12-5-8 is a group mark.

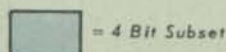
On the other side of a high fence (between scientific and commercial computing at that time) the 704 people come up with FORTRAN, which needs the characters () + and =, and certainly does not need % ◊ & and #. Since there are only 48 positions on the type wheels of the 407, a dual assignment is made. This makes it difficult for the installation with both scientific and commercial problems, but they learn to live with it. Along

Figure 1. American Standard Code for Information Interchange

Example:



	000	001	010	011	100	101	110	111	
0000	NULL	DC ₀	Ⓝ	0	#	P			
0001	SOM	DC ₁	!	1	A	Q			
0010	EOA	DC ₂	"	2	B	R			
0011	EOM	DC ₃	#	3	C	S			
0100	EOT	DC ₄ (STOP)	\$	4	D	T			
0101	WRU	ERR	%	5	E	U			
0110	RU	SYNC	&	6	F	V			
0111	BELL	LEM	'	7	G	W			
1000	FE ₀	S ₀	(8	H	X			UNASSIGNED
1001	HT SK	S ₁)	9	I	Y			
1010	LF	S ₂	*	:	J	Z			
1011	V TAB	S ₃	+	;	K	[
1100	FF	S ₄	(comma)	<	L	\			ACK
1101	CR	S ₅	-	=	M]			②
1110	SO	S ₆	.	>	N	↑			ESC
1111	SI	S ₇	/	?	O	←			DEL



comes the 1401 with its chain printer that has 240 character positions around it, normally in five sets of 48. But it could be in four sets of 60. Because it is to be a satellite machine for many large scale installations, the FORTRAN characters are given their own separate codes for programming convenience. Now we have both dual and individual assignments in the same installation. What confusion!*

Another trouble is that the internal bit assignment for the group mark is 11 1111, and when the code is filled out to the full 64 characters possible it is found that the counting mechanical reader demands that the punch combination for the group mark be 12-7-8, particularly for the 7070 and not 12-5-8 as for the 705. However, the same tape must be capable of being read by both machines.

And so it goes, each mistake by expediency being piled on top of the last one. And so many customers already use these incompatible devices that it just doesn't seem economical to change it now. Or could it be that things will get worse and we will wish we had straightened things out last year before it gets even more expensive?

It should not be thought that IBM is the only manufacturer with such problems. UNIVAC had a similar set of problems, particularly with both 80- and 90-column cards. The RCA 501 was designed with an internal code in which the letters and digits were assigned to consecutive binary numbers, a very sensible arrangement that makes data processing much easier. This is because there is something known as a "collating" or "ordering" sequence. If there were not, it would be very difficult to find a word in the dictionary or a number in the phone book. The 501 orders by simple binary comparison, with no extra hardware or wasted time. If this seems only reasonable, remember that IBM does not make any equipment with an internal code corresponding to its collating sequence, contrary to some beliefs. Ordering is done either by special hardware (\$75 a month for early 1401's) or by programming, as on the 7090. Figure 2 shows two IBM cards, punched and interpreted. Columns 1 to 64 correspond to the binary sequence 00 0000 to 11 1111, or octal 00 to 77.

Let's see what happened to that 501. The 301 was designed to aim at the extensive punched card business. What could be more natural than to forget the 501 code and adopt the internal code of the IBM 704? Later a translator was built to convert codes in both directions between the 301 and 501. Just one problem, though — any file put in order on the 301 was out of order for the 501, and vice versa. At least without programming or additional hardware. This is hardly a trivial problem. IBM calculated in 1961 (in connection with ASA work) that it might take from \$5,000,000 to \$30,000,000 of machine time on the fastest computers just to reorder all existing files (as necessary — most would not require it, having only numeric keys) to a new collating sequence. This is the problem IBM faced in participating in code standardization work. If a standard code were to specify the collating sequence to be identical with the binary sequence, it would not match the IBM collating sequence. Makes even a big company stop and think; it might be hard to get the customer to take the broad view of future advantages and foot the bill.

However, occasions do arise when the situation is so muddled that desperate measures must be taken. As an example, Australia will change over to its new unit of currency, the "Royal", in February of 1966. This will replace the old pound and will be divisible into 100 cents,

(*Author's note—I'll take my share of the blame for some of this.)

which developed and sponsored the Fielddata code. Despite a few drawbacks, this was a great improvement upon existing codes and many of its features are to be seen in ASCII.

4. The British Standards Institution, which also started with the intention of standardizing paper tape codes and

The abbreviations used in Figure 1 mean:			
NULL	Null/Idle	CR	Carriage return
SOM	Start of message	SO	Shift out
EOA	End of address	SI	Shift in
EOM	End of message	DCo	Device control ① Reserved for Data Link Escape
EOT	End of transmission	DC1-DC3	Device control
WRU	"Who are you?"	ERR	Error
RU	"Are you . . . ?"	SYNC	Synchronous idle
BELL	Audible signal	LEM	Logical end of media
FEo	Format effector	So-S7	Separator (information)
HT	Horizontal tabulation	␣	Word separator (blank, normally non-printing)
SK	Skip (punched card)	ACK	Acknowledge
LF	Line feed	②	Unassigned control
V/TAB	Vertical tabulation	ESC	Escape
FF	Form Feed	DEL	Delete/Idle

was drawn gradually into the whole data processing field.

5. The SHARE organization, which sought to coordinate their existing IBM equipment.

major features of the code

Let us examine the several salient and sometimes new features of the code and their significance:

1. As yet it is only a reference code. The particular representations in media such as punched cards, punched tape and magnetic tape are not yet defined, although they are perhaps implied in some respects.

2. It is (so far) a 7-bit code, with provision to expand to 8 bits as required. In the 7-bit form, a 6-bit subset of 64 codes is assigned completely to information characters, the other 64 so far are essentially control characters. This separation can be of convenience to equipment designers in the combined data processing and communications field; it also should produce many economies.

3. Although not yet stated in the standard, there is an implied collating sequence that may be used in the straight binary comparison mode. For IBM, which presently collates digits higher than the alphabet, an Exclusive OR device in passive logic can put the digit vector higher than the alphabet. This is no more than the existing device which allows the 709 family to write and read BCD tape.

4. The set can be collapsed in a regularized and prescribed manner, if required, into a 6-bit set for existing 6-bit machines and other equipments, to a 5-bit set for modification of existing Teletype and Telex sets (particularly in Europe), and even to a 4-bit set. This latter is of very special interest, in that it can be used for cash registers and other basically numeric-only devices, but at the same time may be used in the double numeric mode for computers internally (like the 650, 7070 and STRETCH). It is indicated by the offset shaded vector in the diagram of the code. The reason for the offset is that certain nondigit characters of the 4-bit set must collate lower than both digits and alphabet when appearing in ordering keys. The reverse expansion upward is a simple matter of passive logic.

5. Certain replacements (carefully checked out internationally) allow for non-American usage. For examples, the single digits 10 and 11 (sic) for English pence can replace the colon and semicolon in the digit vector, at least until they follow the lead of the Australians; the characters following the alphabet are of relatively low usage so that they may be replaced with the additional letters of expanded Roman alphabets, particularly as used by the Scandinavian countries.

6. The ESCape code (111 1110) provides for 127 alternate sets in the 7-bit set, 255 in the 8-bit set. Some of these sets may have official standing and some may be arbitrarily reserved to certain equipments. An example might be an alternate set with the Roman alphabet replaced by the Cyrillic alphabet, the unreplaced characters remaining unchanged. The ESCape character is usually followed by another code which is devoid of its usual meaning, by virtue of following the ESCape, and indicates which one of the alternate character sets is in force until the next ESCape character is encountered.

7. The two righthand vectors were purposely reserved as the logical places to put a lower case alphabet, if desired for this to be available in a single character mode. For lesser equipments, the upper case alphabet may be used in conjunction with the Shift In (000 1110) and Shift Out (000 1111) characters to produce a lower case facility.

8. Special consideration has been given for the characters required in programming and other special languages. All of the characters of the COBOL set are included. The ESCape characters may be used to shift to one or more special sets containing all of the characters of ALGOL (including the unique lower case alphabet). Other sets may be reserved for special languages of typesetting, information retrieval, graphic design, medical reports, etc. For example, a special set for numerical machine tool control could be an alternate 4-bit subset which is identical with the standard subset in the thirteen characters 0 - 9, decimal point, plus and minus; the other three characters would be replaced by X, Y and Z for axis symbols.

9. Note that many new characters have been introduced in the control area, particularly designed for self-delimiting of streams of characters. These may be hierarchic in nature, used to describe records, fields, subfields and so forth, or they may be syntactic in nature, indicating phrases, sentences, paragraphs, etc.

why a 7-level code?

Actually ASCII is an 8-level code with the eighth bit unassigned as yet. The new A. T. & T. system, supplied with terminal equipment by its subsidiary Teletype Corp., is based completely on eight bits between start - stop pulses. This is not only for future expansion but also for practical operations today. The eighth bit may be used for parity (preferably odd) if desired, and perhaps other uses may evolve. Basically, however, the 8-bit transmission unit was selected because eight is a magic number, being a power of two. In the information theory business there is nothing more economical than a power of two, and A. T. & T. knows it. Economy is important when you are creating a whole new system of this magnitude, and indeed that magnitude may be well up into the billions. There is even provision for an eleventh digit in the direct dialing system so Teletype can tell whether an 8-bit unit (Model 33) is talking to another 8-bit unit or to a 5-bit unit, or vice versa.

Several new computers are now being designed with 8-bit capabilities. At least one model, STRETCH, is in operation. Another is reported to use ASCII internally in

AMERICAN STANDARD . . .

the double numeric mode. In certain 6-bit machines the word is designed to 48 bits to handle either six 8-bit characters or eight 6-bit characters. This code certainly facilitates transmission of pure binary data.

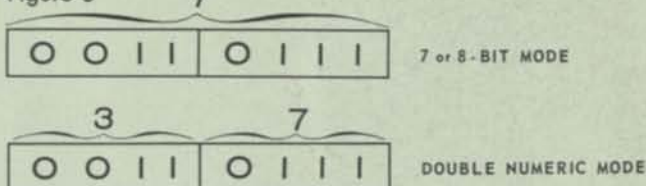
Subcommittee X3.2 appears to have no objection to the eventual assignment of meaning to all of the 256 codes in the 8-bit set. They sensibly avoided trying to be omniscient now and rather made adequate provision for expansion as further developments are made. Besides, they had to consider the Europeans and international standardization work in this area by ISO TC 97 on Computers and Information Processing. This work might not catch up to A. T. & T. for a while. Meanwhile the code will probably have to be adapted to 6-bit systems and even to five bits to work on existing Telex circuits, for the Europeans may not be able to install an entire new system in several countries in less than several years. The ASCII code is certainly set up to reduce the code size as required.

There are many advantages to having more unique codes in the set. There are some still unassigned in the 7-bit set, and of course nothing except a possible parity usage is assigned to the 8-bit set. There are several possible assignments for these spare codes, although none of these have been discussed extensively yet by X3.2:

1. A code which turns parity off and on, or possibly two individual codes, one for each of these two functions. This would facilitate compatibility between equipment using the 7-bit code with parity and other equipment desiring to utilize the full 8-bit set.

2. A code which says "repeat the transmission (it was bad) back to the last S_1 code." Presumably this code would be followed by the particular S_1 code required. This S_1 would be sent back to the transmitting equipment, which would hold it in memory and search backwards along the transmitted stream until a match was found. The transmission would be restarted at that point, both sending and receiving equipment knowing exactly where to pick up again.

Figure 3



3. Codes to ignore normal communications control so that pure binary data may be transmitted without any character meaning. These will have to be handled carefully so that return to the normal transmission mode may be effected. This might have to be done by either timing the binary transmission, sending a predetermined number of 8-bit units with automatic return, or having the receiving device actuate the return through an extra channel.

4. Codes to switch to double numeric (two 4-bit digits within a single character) and back for reasons of economy of transmission in numeric only mode. ■

(Part two of Mr. Bemer's article will be published next month.)

You Get Things Done Better By Seeing What's Happening



BOARDMASTER VISUAL CONTROL

- ★ Gives Graphic Picture of Your Operations in Color.
- ★ Easy to Use. Type or Write on Cards, Snap on Board.
- ★ Facts at a Glance—Saves Time and Prevents Errors.
- ★ Ideal for Production, Scheduling, Sales, Inventory, Etc.
- ★ A Simple, Flexible Tool—Easily Adapted to Your Needs.
- ★ Compact, Attractive. Made of Metal. 750,000 in Use.

Complete Price **\$49⁵⁰** Including Cards

FREE

24-Page ILLUSTRATED BOOKLET CG-20
Without Obligation

GRAPHIC SYSTEMS, Yanceyville, North Carolina

CIRCLE 25 ON READER CARD

THE CAT COMPUTER

■ Already enjoying wide acceptance without formal announcement is a multi-purpose on-line CAT Computer of Average Transients. One at Mayo Clinic has participated in a transatlantic experiment, averaging out brain wave signals transmitted from England via the Relay satellite. The results, interpreted and diagnosed, were then sent back by the same route. Present applications are in medical and clinical research.

The CAT 400B is a product of the Mnemotron Div. of Technical Measurement Corp., White Plains, N.Y. The count capacity of the memory is 100K per ordinate, with up to 400 ordinates. The CAT can sum and average responses from four varying inputs on-line, simultaneously calculating and displaying data on a built-in scope. The ability to store averages in successive quarters of memory makes it possible to compare successive runs of averages without interrupting experiments (there is no theoretical limit to the number of responses which may be summed). Averages may be displayed on the 3" CRT and on an X-Y Plotter Readout. The computer measures about one cubic foot, weighs 38 pounds, and consumes 30 watts. Price is \$12K. ■

THE AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE

by R. W. BEMER, UNIVAC Division, Sperry Rand Corp., New York, N.Y.

Synopsis—In the first of this two-part article, Mr. Bemmer covered the inglorious history of information coding which led to the ASCII, becoming an official ASA Standard No. X3.4 on June 17, 1963. He also covered its salient features, and explained the seven-bit code with provisions to expand to eight bits.

the conversion problem

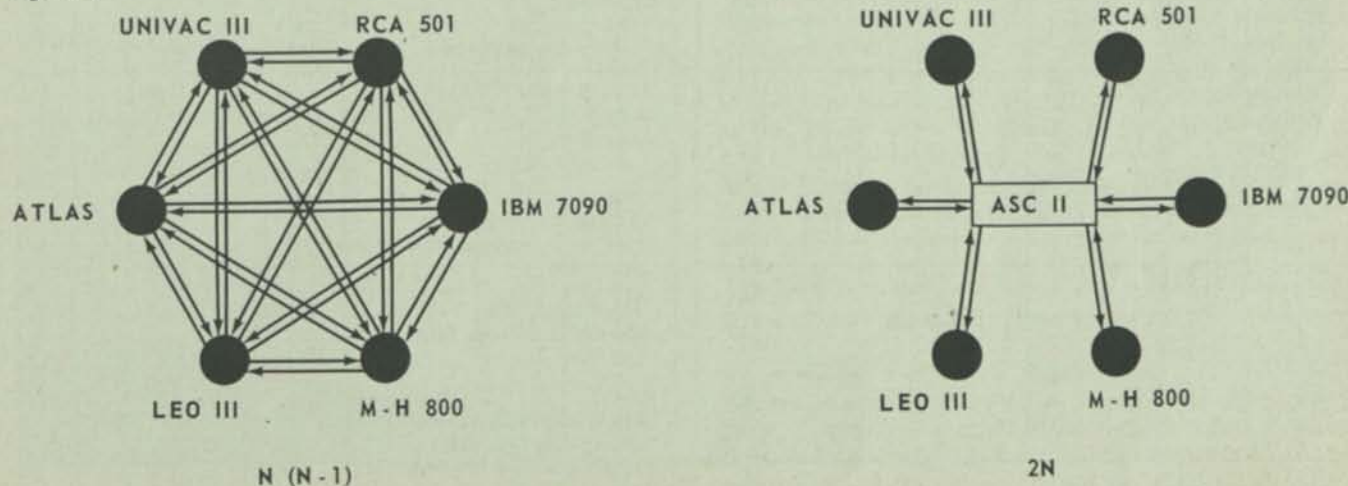
The major argument against the new code seems to be the cost of converting the vast amounts of equipments and customers, with resulting obsolescence (at least of the equipment). The intent is apparent in the title of the standard — "for INFORMATION INTERCHANGE." This does *not* say that computers of external devices must be built to use this code internally, now or ever. All it demands is that whenever the computer talks with strange equip-

ment, *not of its own kind*, that it do so through the medium of this code.

Certainly this results in fewer translation mechanisms than the present chaotic situation requires. Given N computers or other devices with various and different internal codes, each might need to talk with all the others ($N-1$). Thus N times ($N-1$) translations would be required for full intercommunication. With ASCII, however, each device needs to talk only to the standard code and back again, a total of only $2N$ translations required! The value of N is presently about 60 (internal codes). Although one would hardly expect that all possible 60 times 59 combinations would be used, it is certainly enough larger than 60 times 2 to say that even if every present day machine retained its own code forever, it would still be more economical to use ASCII just for interchange! Thus each machine would have to talk only to ASCII instead of 59 other codes. (See Figure 4.)

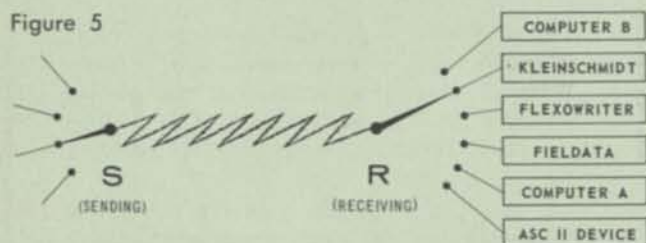
Furthermore, the possibility should not be overlooked

Figure 4



that some other internal code *plus* the translation mechanisms required might be more economical for some equipment than would ASCII internally. Of course, the economic pressure for future equipments to use the code internally as well as externally would be more likely. Thus the 2N combinations might even be reduced to *one* some time in the future. The new code has so many inherent economies that it might pay for the redesign itself. IBM has perhaps the least problem of any manufacturer; with 9 different codes already in their various computers, ASCII presents only an 11 per cent additional problem.

During the conversion period, the ESCape character allows most existing codes to exist simultaneously with ASCII. Assume a communications link as in Figure 5. When a message comes along that is not in ASCII, the first character is ESCape, the next is that which selects the alternative code. The message in alternative code then follows until the next ESCape character signals either another alternative code or return to ASCII. Physically the receiving terminal R will be alerted for switching by



ESCAPE; the following character actually performs the switching to different receiving equipment for the alternative code. This concept is much simplified, of course, and in practice might be applied only to long distance links.

It would seem a practical thing for X3.2 to assign a block of codes (to follow ESCape) to indicate the codes of existing equipments. As these become obsolete, the particular selector code may be reassigned for other purposes.

advantages for programming

If ASCII were to be built in as an internal computer code, the programmer might expect to see some of the following benefits:

1. Manipulation of graphics by classes. Since all characters of a certain class (such as letters, digits, etc.) are grouped contiguously, they may be classified with very few instructions. In working with strings it may be useful to create a corresponding class string in parallel for syntax analysis. This could open some interesting doors in library work and information retrieval in general.

2. Fewer instructions in scans, due to regularity and unique codes. A count once made of 709 FORTRAN showed that something like 53 instructions were required to decipher the syntactical meaning of a left parenthesis. As no other brackets were available, parentheses were used for subscripting, normal mathematical nesting and other purposes. With unique codes, the combination can form the address of the starting instruction of the routine for processing that character. Consider that the IBM FORTRAN market amounts to about \$150,000,000 a year in just machine time used. The figure commonly accepted for translation from FORTRAN to machine language is 35 per cent of total time. Thus about \$50,000,000 a year is spent on nothing but FORTRAN translation. Certainly

OCTAL CODE	ASCII GRAPHIC	IBM				UNIVAC	UNIVAC				RCA
		SIC A	SIC H	COLL. SEQ.	7090 CORE		1050 A	1050 H	490	1107	
00		00	00	00	00	05	00	00	05	05	00
1	!	52	52	43	52	55	43	43			
2	"					57					05
3	#	13		24			35	37	04		36
4	\$	53	53	07	53	47	42	42	47	47	07
5	%	34		17			61	17	52		10
6	&	60		06			20	63	46	46	12
7	*		14	25	14	72	15	56	72	72	13
10	!		34	17	74	51	17	61	51	51	
1	!		74	02	34	40	01	75	40	40	04
2	*	54	54	10	54	50	41	41	50	50	15
3	+		60	08	20	42	53	20	42	42	
4	-	33	33	16	73	56	62	62	56	56	33
5	.	40	40	14	40	41	02	02	41	41	14
6	.	73	73	01	33	75	22	22	75	75	16
7	/	21	21	15	61	74	64	64	74	74	22
20	0	12	12	66	00	60	03	03	60	60	23
1	1	01	01	67	01	61	04	04	61	61	24
2	2	02	02	70	02	62	05	05	62	62	25
3	3	03	03	71	03	63	06	06	63	63	26
4	4	04	04	72	04	64	07	07	64	64	27
5	5	05	05	73	05	65	10	10	65	65	30
6	6	06	06	74	06	66	11	11	66	66	31
7	7	07	07	75	07	67	12	12	67	67	32
30	8	10	10	76	10	70	13	13	70	70	33
1	9	11	11	77	11	71	14	14	71	71	34
2	:	15	15	26		53	21	21	53	53	06
3	;	56	56	12		73	16	16	73		11
4	<	76	76	04		43	36	36			43
5	=		13	24	13	44	37	35			44
6	>	16	16	27		45	76	76			45
7	?	72	72	31	32	54	23	23			

SIC - STANDARD INTERCHANGE CODE

OCTAL CODE	ASCII GRAPHIC	IBM				UNIVAC	UNIVAC				RCA
		SIC A	SIC H	COLL. SEQ.	7090 CORE		1050 A	1050 H	490	1107	
40	@	14		25			56	15			
1	A	61	61	32	21	06	24	24	06	06	40
2	B	62	62	33	22	07	25	25	07	07	41
3	C	63	63	34	23	10	26	26	10	10	42
4	D	64	64	35	24	11	27	27	11	11	43
5	E	65	65	36	25	12	30	30	12	12	44
6	F	66	66	37	26	13	31	31	13	13	45
7	G	67	67	40	27	14	32	32	14	14	46
50	H	70	70	41	30	15	33	33	15	15	47
1	I	71	71	42	31	16	34	34	16	16	50
2	J	41	41	44	41	17	44	44	17	17	51
3	K	42	42	45	42	20	45	45	20	20	52
4	L	43	43	46	43	21	46	46	21	21	53
5	M	44	44	47	44	22	47	47	22	22	54
6	N	45	45	50	45	23	50	50	23	23	55
7	O	46	46	51	46	24	51	51	24	24	56
60	P	47	47	52	47	25	52	52	25	25	57
1	Q	50	50	53	50	26	53	53	26	26	60
2	R	51	51	54	51	27	54	54	27	27	61
3	S	22	22	56	62	30	65	65	30	30	62
4	T	23	23	57	63	31	66	66	31	31	63
5	U	24	24	60	64	32	67	67	32	32	64
6	V	25	25	64	65	33	70	70	33	33	65
7	W	26	26	62	66	34	71	71	34	34	66
70	X	27	27	63	67	35	72	72	35	35	67
1	Y	30	30	64	70	36	73	73	36	36	70
2	Z	31	31	65	71	37	74	74	37	37	71
3	[75	75	03			55	55			
4	\	36	36	21			40	40			
5]	55	55	11			77	77			
6	^										
7	_										

all of this is not due to the left parenthesis problem, but it ought to run to at least a million.

3. Faster and cheaper sorting, when the collating sequence is identical to the binary sequence of the codes for the graphics. Sorting is also big business, with commercial users quoting an average of 40 per cent of total machine time used for this one function. The elimination of special hardware for comparisons would save more than a million dollars a year.

4. Reduction in the number of routines required to be programmed, particularly for satellite equipment. The chart of Figure 6 indicates the complexity of routines that must be provided for a multiplicity of codes. The ASCII code is taken as the base code in binary sequence. The corresponding octal codes for the same graphics are given for the various other internal codes. Obviously the same procedure could be followed using any particular code as the base code. The totality of such charts provides the basic information for generalized code conversion among various equipments.

5. Fewer tables for mixed codes in communications, particularly those controlled by store-and-forward message switching systems. An IBM spokesman stated that the 7750 communications unit rents for \$8,000 a month with a single code, up to \$13,000 a month to handle all codes, since additional core storage is required for programs and tables to handle these other codes.

6. Clarity of printed output, particularly the reproduction of the source program in the printed record of processing. Unavailability of the exact graphic desired makes for costly mistakes in the diagnostic process. It takes quite a bit of practice to get used to reading FORTRAN with the per cent sign and lozenge used instead of parentheses.

7. A tendency for keyboards to be identical with typing communications equipment. Thus hard copy can be available immediately as a record of the program being key-punched. It is conceivable that this might extend to halflines spacing for subscripts and superscripts, a feature which might have a considerable effect in relaxing restrictions in the rules of programming languages.

the future for the ASCII code

X3.2 is presently going full steam ahead in implementing the code in the various media. This will not be a simple problem, particularly in punched cards. Presumably the binary code could be duplicated directly, a punched position standing for a 1, an unpunched position standing for 0. But there are 12 positions on the card, not 8, and that is a little wasteful. Besides, certain punching equipment will not perform up to specification when punching more than three or four holes in a column. It is possible to represent 256 codes by combinations of 0, 1, 2, and 3 punches (and no more), but this is not easy if it is required to make the combinations consistent with present punched card practice. A difficult problem, surely, but a look at the references following this paper will indicate that much work has already been done.

What will happen now to other contenders? It seems clear that Fieldata, even though implemented already in many computers, will gradually be replaced by ASCII. Indeed, Fieldata representation on X3.2 was very strong and valuable. Fortunately the Department of Defense is committed to national and commercial standards wherever they exist, even in preference to some military standards, and so Thomas Morris, Assistant Secretary of Defense, has been instrumental in the completion and adoption of the ASCII code.

It is not likely that the code will be adopted internationally in the exact form that it is in now. However, the

William Orchard-Hays

and David M. Smith

Announce

the Organization of

ORCHARD-HAYS & COMPANY, INCORPORATED

Two of America's most widely recognized experts in engineering large-scale systems of programs—William Orchard-Hays and David M. Smith—have brought together a select group of other software specialists to establish Orchard-Hays & Company, Inc.

Orchard-Hays & Company has a particularly broad capability in linear programming, having engineered highly-advanced LP systems for the IBM 7090 and 7094 computers. For other computers, O-H&C will implement its advanced design techniques in LP systems ranging in scope from the most basic to the most advanced. Systems can be adapted to virtually any hardware configuration and can be easily and economically upgraded to provide greater problem-solving power and flexibility as your needs broaden.

Orchard-Hays & Company also creates specialized information handling systems, statistical systems, specific-function computer languages, program processors, and other application systems, as well as undertaking program conversion when computer hardware is replaced with more advanced machines.

O-H&C software is engineered for total efficiency and utility. Program systems engineering is the first order of business at O-H&C, not a sideline to computer time sales or other activities. No matter how complex your program system requirements may be, we are confident that O-H&C know-how will provide the answer. Call us collect.

O-H&C

ORCHARD-HAYS & COMPANY INCORPORATED

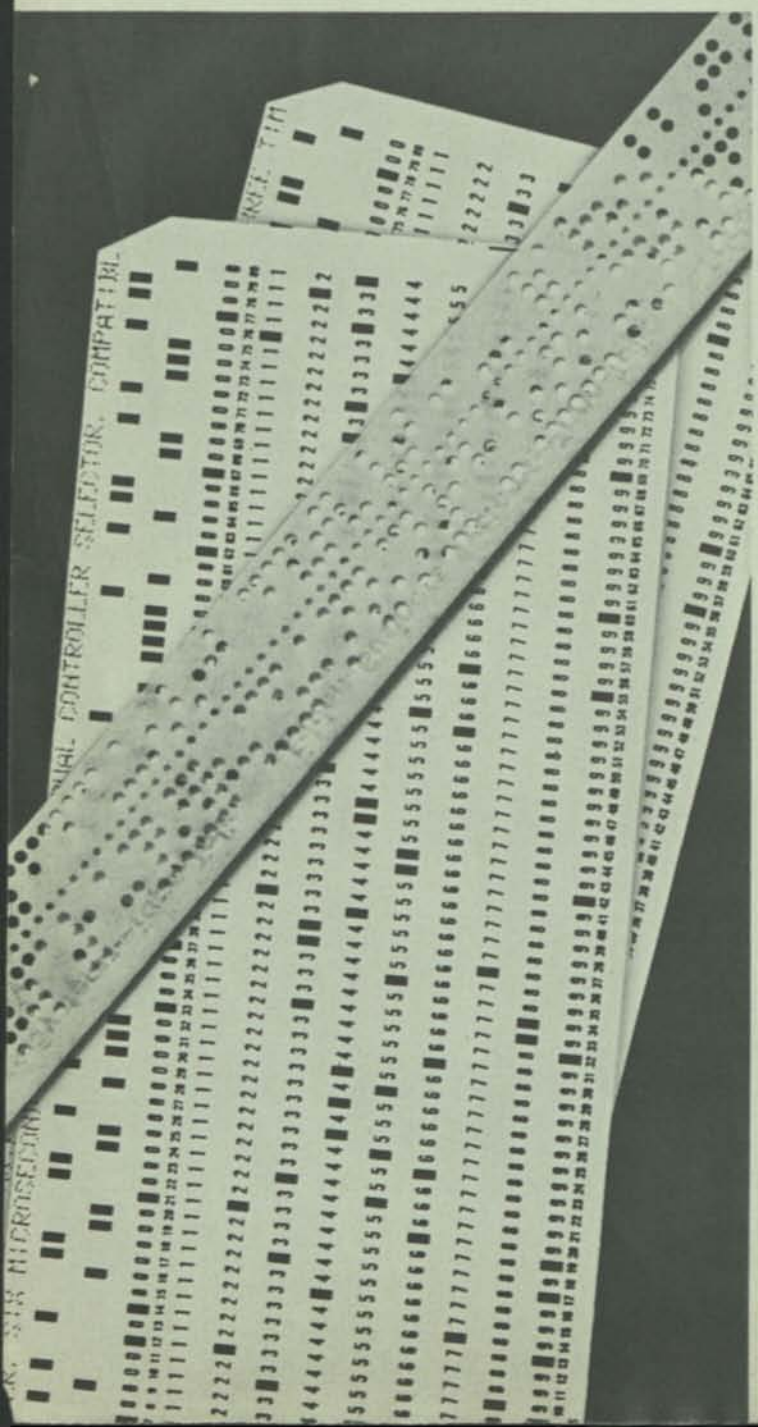
3150 Wilson Boulevard, Arlington 1, Virginia, 22201
Area Code 703—Phone 525-5206

THE COM

General Electric's new family of computers lets you

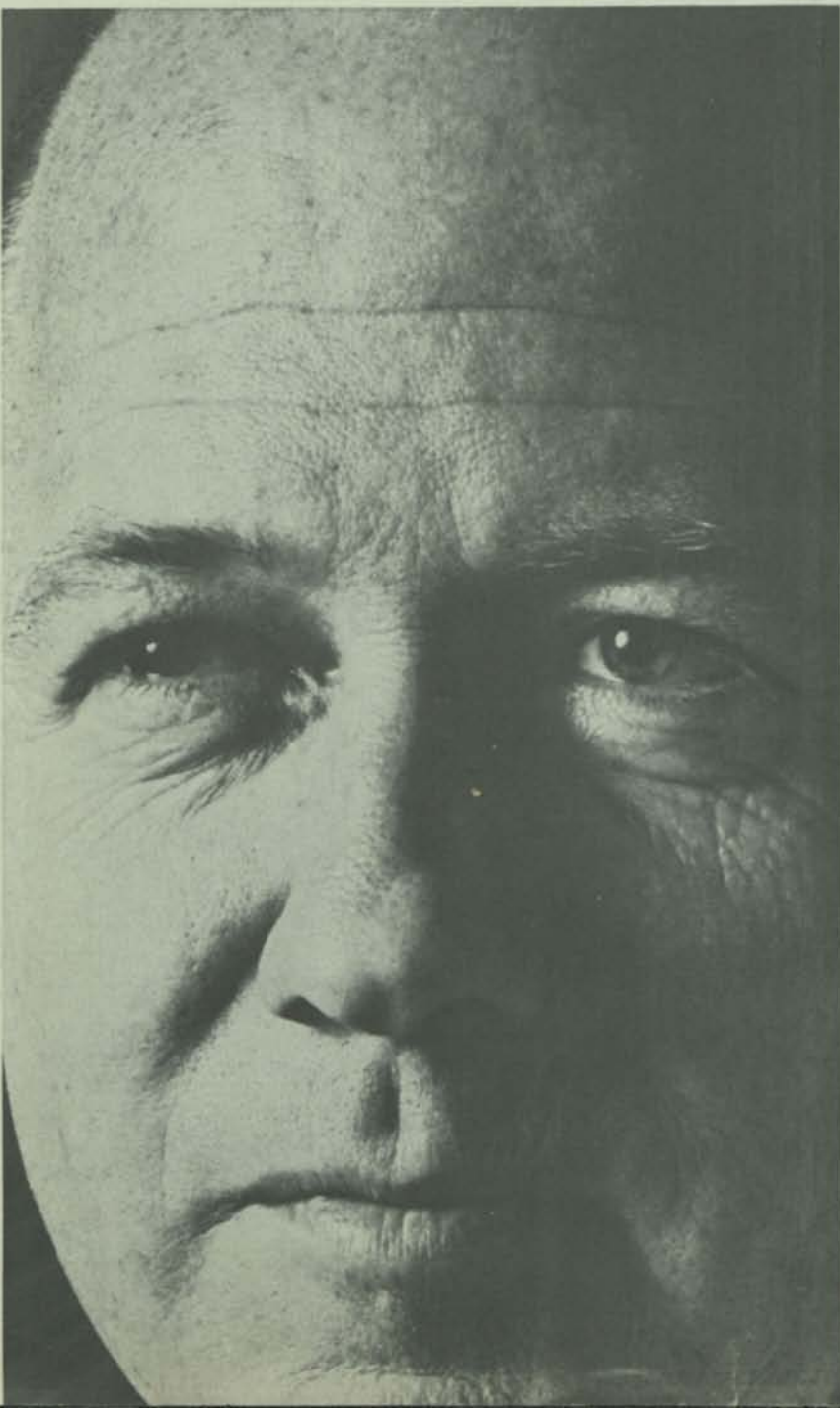
programs...

peripherals...



PATIBLES

increase your computing power without changing...
or people



You can move up from a minimum cost system to bigger, faster systems by simply changing the central processor. The programs are the same—but they run faster. The peripheral equipment is the same—only you can apply it more efficiently. The people are the same ones you trained in the first place.

So, before you buy or replace your system, investigate The Compatibles. Write General Electric Computer Department, Section J-9, Phoenix, Arizona.

GE-215

GE-225

GE-235

Tops in its price range: 36 microsecond word time	Medium-sized, versatile system: 18 microsecond word time	Most powerful in the family to date: 6 microsecond word time
--	---	---

Progress Is Our Most Important Product

GENERAL  ELECTRIC

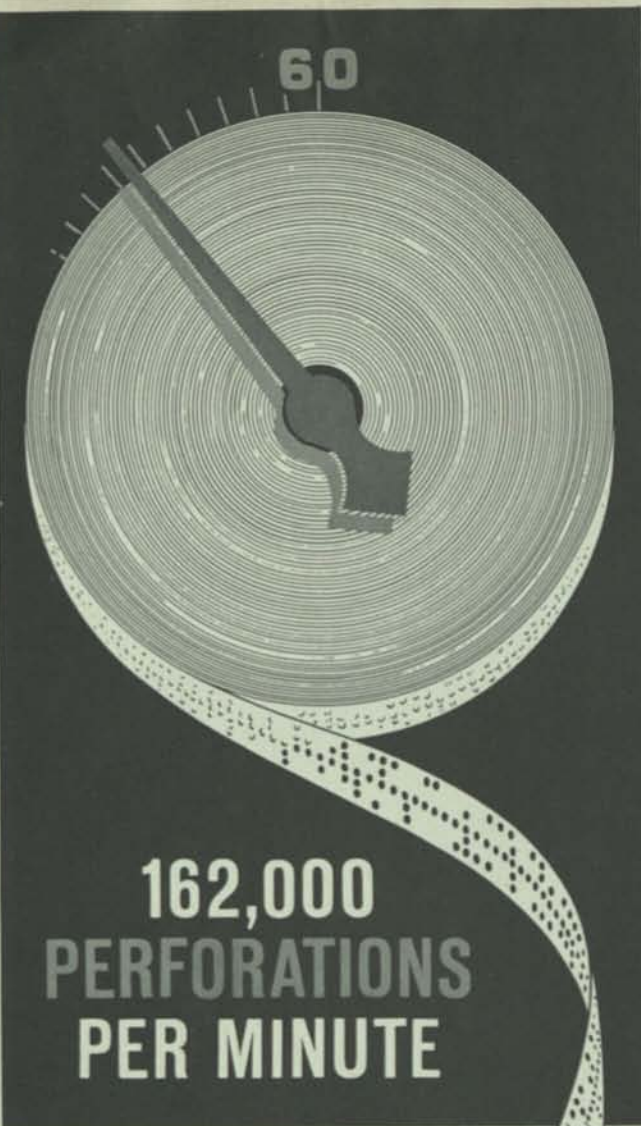
CIRCLE 23 ON READER CARD

U.S. has taken considerable pains to meet international requirements and plan ahead. It is likely that a closely related code will become an international standard, in which case the ASA must simply make some modifications. Don't forget, standards are not cast permanently in bronze; they must adapt to the time and circumstances. Actually the degree of cooperation with international standards bodies and consideration of their requirements has been rather a milestone in the case of this code and other computer standards. Formerly the U.S. has either ignored or minimized international requirements. In the case of ASCII the international implications are such (satellite transmission, etc.) that full cooperation was mandatory, if only for the mundane reason that if another war were ever fought in Europe it would be a considerable advantage to be able to use existing communications equipment.

It is also a fact that the computer and information processing market outside of the U.S. is expanding greatly, and U.S. manufacturers must consider the expense of rebuilding such costly things as computers to match non-U.S. standards. It may be that the Russians will ignore this code, even though their requirements have been considered. My guess is that economic motivations of a less controlled society will win again in the American Standard Code for Information Interchange. ■

REFERENCES

1. British Standards Institution, Alpha-Numeric Punching Codes for Data Processing Cards, British Standard 3174:1959.
2. British Standards Institution, Committee DPE 149, Draft British Standard for Punched Tape Coding, Part 1, 7 Track Code, AA (DPE) 3543, September, 1960.
3. Electronic Industries Association, EIA Tentative Standards Proposal for Eleven-sixteenths Inch Wide Five-Track Perforated Paper Tape.
4. Electronic Industries Association, EIA Standards Proposal for One Inch Perforated Paper Tape.
5. Electronic Industries Association, Committee TR24.4, Language and Format For Punched and Magnetic Tapes, Document 7233, May, 1960.
6. U.S. Army Signal Corps, Fielddata Equipment Intercommunication Characteristics, memorandum for: Director, Data Processing, Facilities Division, Communications Department, USA-SRDL of April, 1959 (Revised August, 1959).
7. U.S. Department of Defense, Military Standard 188A (Fielddata)
8. Bemmer, R. W., A Proposal for Character Code Compatibility, *Communications of the ACM*, February, 1960.
9. Buchholz, W., An Extended Character Set Standard, IBM Technical Report TR 00.721, June, 1960.
10. _____, A Survey of Coded Character Set Representation, *Communications of the ACM*, December, 1960.
11. Smith, H. J., Jr., and Williams, F. J., Jr., Design of an Improved Transmission/Data Processing Code, *Communications of the ACM*, May, 1961.
12. Luebbert, W. F., Information Handling and Processing in Large Communication Systems, Tech. Report 099-1, Stanford Electronics Laboratories, Stanford University, 11 July 1960.
13. Ross, H. McG., Considerations in Choosing a Character Code for Computers and Punched Tapes, *The Computer Journal*, January, 1961.
14. _____, Further Survey of Punched Card Codes, *Communications of the ACM*, April, 1961.
15. Smith, H. J., Jr., A Short Study of Notation Efficiency, *Communications of the ACM*, August, 1960.
16. _____, and Williams, F. A., Jr., Survey of Punched Card Codes, *Communications of the ACM*, December, 1960.
17. *Business Week Magazine*, 15 June 1963, pp 128-131, Versatile Passkey to Communication.



Only the very finest tape is good enough for today's newer, super-speed, tape perforators!

When you specify PERFECTION®, you know you're getting the finest . . .

holes are clean and sharp, down to the last perforation . . . tapes don't tear or break, even at the highest speeds . . . slitting is sharp and true, and virtually lint-free . . . base stocks are chosen-for-the-job, and quality assured.

PERFECTION® Tapes, either rolls or folded, are available for every computer or communication application. Write today for a sample brochure and the name of your nearest PERFECTION® Distributor.

PAPER MANUFACTURERS COMPANY

PHILADELPHIA 15, PENNSYLVANIA

BRANCH FACTORIES:
Indianapolis, Ind. • Newark, Calif.

SALES OFFICES:

Atlanta • Chicago • Cleveland • Dallas
Kansas City • Los Angeles • New England
New York • San Francisco • Syracuse

PERFECTION®

Additional copies of these proceedings may be obtained at a cost of \$2.00 (plus tax).

IIT Research Institute
10 West 35th Street
Chicago, Illinois 60616
Attention: Sales
Telephone: (312) 941-1000
Teletype: (312) 941-1000

PROCEEDINGS OF THE JUNE, 1964

APT

TECHNICAL MEETING

CHICAGO, ILLINOIS



IIT RESEARCH INSTITUTE

Additional copies of these proceedings may be obtained at a cost of \$5.00 from:

IIT Research Institute
10 West 35th Street
Chicago, Illinois 60616

Attn: Main Files

©
Copyright, 1964
IIT Research Institute
Chicago, Illinois

NUMERICAL CONTROL AND PUBLIC COMPUTING POWER

R. W. Bemer
UNIVAC Division
Sperry Rand Corporation

Abstract

The greater expansion in numerical control usage will come from the smaller customer with access to large scale computing facilities. In most cases this smaller customer will be unable to justify a large computer initially; he has the alternatives of traveling to a central service bureau or sharing time on a large computer via communications devices operable from his own place of business. It is the contention of this paper that all the requisite hardware and software techniques are available now for creation of public computing power facilities. This complex can service many other industries in addition to numerical control users. APT will be but one language spoken by the user of public computing power; FORTRAN, COBOL and other specialized languages will be utilized to share such facilities. The purpose of this paper is to present some initial logistical methods for economical exploitation of such a facility, with the hope of creating practices of such broad application that they can become the basis for standards in this new field.

1. Origins of Various Elements Involved

The computer is often said to amplify man's brainpower in much the same manner as machines amplified his muscle power. Many power sources are public utilities. Is there any reason why computing power should not be similarly available to the general public? Except in remote places it is uneconomical for one to own a complete power plant. This also applies to computers. All elements of hardware and software are now available for the establishment of Public Computing Power. The systems concepts have existed for some time. To illustrate, an excerpt is included here from an article of mine which appeared in the March 1957 issue of Automatic Control Magazine, reprinted with the permis-

sion of the editors. Perhaps my emphasis on microwave has not been matched by present developments, but the essentials are there.

Future Computer Systems

Future computer operation, which strongly influences the design of the programming languages, has some vitally interesting possibilities. In this glimpse, the picture presented here is dependent upon three axioms:

- Faster computers always lower the dollar cost per problem solved, but not all companies will be able to afford the high prices of the next generation of super-computers. They simply may not have enough problems to load one.

- Producing a spectrum of machines is a tremendous waste of effort and money on the part of both the manufacturers and the users.

- Availability of a huge central computer can eliminate the discrete acquisition of multiple smaller computers, homogenize the entire structure of usage, and allow a smaller and more numerous class of user into the act, thus tapping a market many times the size of presently projected with current practice in computer access.

Assuming the availability of practical microwave communication systems, it is conceivable that one or several computers, much larger than anything presently contemplated, could service a multitude of users. They would no longer rent a computer as such; instead they would rent input-output equipment, although as far as the operation will be concerned they would not be able to tell the difference. This peripheral equipment would perhaps be rented at a base price plus a variable usage charge on a non-linear basis. The topmost level of supervisory routine would compute these charges on an actual usage basis and bill the customer in an integrated operation. These program features are, of course, recognizable to operations research people as the Scheduling and Queuing Problems.

Using commutative methods, just as motion pictures produce an image every so often for apparent continuity, entire plant operations might be controlled by such super-speed computers.

These future hardware capabilities (and few competent computer manufacturers will deny the feasibility, even today, of super-speed and

interleaved programs) demonstrate a pressing need for an advanced common language system so all users concerned can integrate their particular operations into the complex of control demanded by an automated future.

The elements of communication necessary for PCP were:

- a. A spectrum of inexpensive terminal devices connectable to existing switching networks.
- b. Expansion from the limiting Baudot five-track code to an eight-bit code for public communication facilities, as evidenced by the changeover of the Bell and AT&T systems to the ASCII (American Standard Code for Information Interchange).

The elements of language necessary to use PCP were:

- a. Specialized languages which had to be comprehensive, powerful and machine-independent to a considerable extent. APT is one of these; FORTRAN and COBOL are examples of others.
- b. Specialized languages for lexical processing and composition, such as the work of M. P. Barnett at MIT. These are required for remote manipulation of source documents for change, correction, deletion, insertion and copying in various ways.

2. The Market for Public Computing Power

These several factors contribute to building a demand for public computing power:

- a. The often rediscovered fact that the larger and more expensive the machine, the cheaper it is to do a given problem.
- b. The discrete nature of a physical computer. You may have none, one, two or more—never a part of a computer, unless you share it with someone.
- c. The high rate of obsolescence of commercially available computers (5 additions per second in 1949 to 500,000 in 1962). It is less expensive to upgrade fewer larger machines.
- d. The widening sector of people who know how to process problems for computers via the special languages. In particular,

FORTRAN is taught now in almost all major universities. APT III may also expect to receive this treatment.

- e. The desire to have work done on demand, and without the bother of maintaining a computer installation.

Numerical Control will be the fastest-growing market for PCP. We are told that there are fewer than 5,000 N/C tools in existence today, and that a major portion of our machine population is due for replacement. Taxes on inventory provide a strong impetus. However, this group meeting today would not exist without the symbiosis between the computer and the machine tool, and that symbiosis is in imbalance in that one computer can serve more than a thousand tools. Therefore N/C will not achieve maximum growth and profit without PCP.

On the other hand, N/C alone cannot be expected to foster and pay development costs for such a complex network. Fortunately there are many other potential users—universities, small businesses, the armed services and even the corner drugstore. PCP will be useful for a variety of applications, which could include composition for technical publications, newspapers and books, language translation, traffic control, linear programming, transportation models, shipping and inventory, accounting, project control thru PERT/COST, scientific and engineering problems, and many others. Perhaps there will one day be a unit of Public Computing Power corresponding to the kilowatt, and the more you use the cheaper it will be.

3. PCP Hardware

The heart of the system must be a general purpose computer with at least the following features:

- a. Realtime capability and Externally Specified Interrupts (i.e., the unit demanding service must leave identification and a means to continue contact).
- b. Concurrent operation, the ability to run several programs at least interleaved and perhaps simultaneously.
- c. Lockout for protection of the segments of store in use by a customer, and scrambling features for security.
- d. Sufficient clocking and indicator mechanism to be able to ac-

count for the usage of each element of the computer on a single job.

- e. High reliability and virtually no downtime. This might be accomplished either by multiplexing or by utilizing idle time on various components to exercise reliability tests and verify ability to respond to demand.
- f. Plenty of input-output channels to both peripheral equipment at the center and to communications lines terminals.

The terminal equipment must be modular and matched as to interface. It must be capable of offline operation to do useful work independent of the central computer. Hard copy must be produced when originating data, and when receiving output from the computer. Paper or magnetic tape are suitable storage media. The punched card will lose ground consistently. US usage has been mainly with cards for editing flexibility, the European usage has been mainly with paper tape for economy and they have forced themselves to prepare perfect copy. With a computer online, corrections do not have to be made in place; they can be described further down the tape and the computer can do the correction and editing during the necessary scanning process. Terminal equipment which meets these requirements is now in production.

With a Teletype Model 35 terminal and a UNIVAC 1004 one can run a remote computer with such facility as to fool the casual observer into thinking the whole computer was miniaturized in those units.

The last element of hardware is the communication system itself. Both public and private lines should be available. The most necessary condition is that the system can be operated in a code-insensitive mode as required.

4. PCP Software

Fortunately software can be modified more flexibly than hardware, for there will be much to learn. There appear now to be four main elements in a comprehensive executive system which controls all processors identically, regardless of source or demand:

- a. Priority routines which react thusly:
 1. Immediate—calls processor as soon as feasible among demands of equal priority for other processors. Processes

and returns results as soon as lines are available.

2. Normal—notes request and starts clock for that processor. Calls processor either after predetermined maximum elapsed time or after minimum number of requests for use (whichever is earliest).
 3. Two hour or overnight— schedules usage of various processors under its control to best utilize available facilities and still rotate testing of components to maintain oncall capability.
- b. Accounting and billing routines which compute charges according to priority of service and usage of components. They verify authority for charging a service, to protect against bootlegging or mischarging. They prepare monthly bills and either send unsolicited monthly TWX messages to each customer or a mailed bill, or both.
 - c. Routines for utilizing mass storage for stocking of course programs and translations. They will retrieve previous programs for change or cannibalizing, perhaps by more than one user if copyright is waived. They log usage and periodically rearrange the storage pattern in levels based upon frequency, for minimum turnaround time.
 - d. Editing routines to make perfect copy from copy submitted by customer which uses downstream corrections. They accept patches upon option, rework and submit for reprocessing.
5. Logistics for PCP

The Model 35 TWX seems ideal for N/C usage at this time. It may be used offline as a typewriter or for preparation of programs and data. Online, as a member of a standard network, it may be used as an inquiry station, for reservations, for ordering and billing, and for Information Retrieval. For Numerical Control applications let us first consider the semi-online method via corresponding terminal equipment at the computer center. Direct communication of the remote terminal will become more practical later.

Imagine the parts programmer in the factory writing his program on the TWX in the local mode. When he believes he is ready he dials a

Data Processing Center and requests service. Table 1 shows the projected formats for the various tapes which will be created. Note that in every case there are three main information sections prior to the working program data. The first section on dial-up is used for any remote service. The second section identifies the usage of TWXOL (TWX Oriented Language) and makes further branches to identify more specifically what is to happen. The third section gives details of actions to be taken and options selected.

Very probably the program will contain mistakes the first time it is processed. For this reason it is not advisable to request either a CL Tape or an object tape. After the user dials the center his tape is sent and duplicated on the center TWX. The operator then puts this on the paper tape reader of the computer, which immediately reads it to secondary store on a concurrent basis, noting the request, its priority and class of service. After processing the computer produces output tape, probably with several outputs adjoined for various customers. The center operator then dials the customer and sends his diagnostics, plus the results of any other option requested. This then becomes a re-cycling process with the successive reduction of syntactic, semantic and pragmatic mistakes. When it is believed that all mistakes are corrected, the parts programmer may request a CL or object (part) tape, presumably to try to fabricate a part. This is where we find the mistakes which are not possible for the computer to detect.

When the source program is lengthy the user will likely elect to pay something to have it kept in the secondary store of the computer. In this case he uses the editing facilities of TWXOL to update his source program. If the return tapes are lengthy he may elect the patching option.

Returning a patch tape(s) to either a source program, a CL Tape, or object tape may be dictated by economy of transmission time, and is at the option of the requestor. The format of a patch tape and the schematic of the patch process are shown in Figure 1. The dimensions A and B are given by the initial message "patch starts A inches, extends B inches." A and B are given to a precision of .1 inches, although this is probably not necessary in actual practice. B is never less than 2" for physical manipulation and splicing, regardless if some of this area is still correct. Multiple patches may be sent in the same tape.

The patch is determined in the computer either by matching the new version to the old (which may be expensive in machine time) or by indications of logical breakpoints which are either furnished by the user in the language or computed as part of the translation process. This indicates that there must be an exclusive one-to-many correspondence of source statements to object code. Obviously in order to use a patch option the user must have previously identified an original program or CL Tape and specified a save option for mass store. If further messages are required, the general form (not necessarily limited to patching) is:

1. Delete string
2. "So and so follows delete string"
3. Delete string
4. So and so

The patch will usually have been printed during transmission. The user marks the previous tape approximately, using dimensions A and B. The 3" overlap section is moved back and forth around this point to obtain an exact and full match. The mark is corrected to the very character and the tape cut and spliced with the delete string following as the splice part. The same process is applied to the other end of the patch.

As justification for making such an option available, our estimate for each test part for this system is 5000 characters of input vs. 26,000 characters of punched tape and printed output. For one cycle, line costs are estimated at \$12.25. This is for a correct program. For many passes thru the computer (for corrections and diagnostics) this would be more expensive than the human time required to patch. This method may be unnecessary if Data Speed units are utilized.

Table 2 is a composite of the logical options which the parts programmer may request. The computer program will check the validity of the specified options with respect to minimums and logic combinations. The asterisk indicates that if the tape is also returned by TWX the mail copy should be computer verified by either duplexing or checksum. This applies primarily to CL and object tapes. Many other checks are performed in the cyclical correction process by matching to the last copy contained in the secondary store.

6. Requirements for Startup

There are several features which must be added or amended to

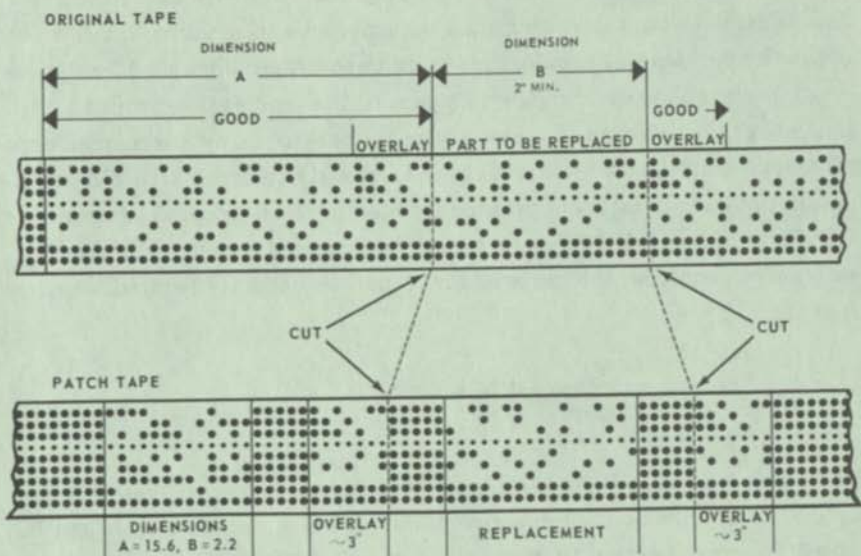


Figure 1

Figure 1

ACTION	ITEM	TAPE		HARD COPY		OPTION TO KEEP IN MASS STORE
		TWX	MAIL	TWX	MAIL	
SEND TO CENTER	SOURCE PROGRAM	OK	OK	AUTOMATIC	OK	OK
	SOURCE PROGRAM PATCH	OK	NOT USEFUL	AUTOMATIC	OK	UPDATED
	CL TAPE	OK	OK	NOT CLEAR	NOT USEFUL	OK
EXECUTION PRIORITY	1. IMMEDIATE					
	2. NORMAL					
	3. 2 HOURS					
	4. OVERNITE					
OUTPUT FROM CENTER	DIAGNOSTICS	NOT USEFUL	-	OK	SLOW	-
	UPDATED SOURCE PROGRAM	OK	OK*	(AUTOMATIC)	OK	OK
	UPDATED SOURCE PROGRAM PATCHES	OK	-		-	-
	CL TAPE	OK	OK*	(NOT CLEAR)	-	OK
	CL TAPE PATCHES	OK	-		-	-
	OBJECT (PART) TAPE	OK	OK*	(NOT CLEAR)	-	OK
	OBJECT (PART) TAPE PATCHES	OK	-		-	-

Table 2

make PCP practical and economical. Among these are:

- a. Strict adherence to ASCII code in all possible equipment and media. I strongly urge that manufacturers of numerically controlled equipment using the EIA standard code provide plug-boards during the conversion period, so that ASCII tape is actually used to control the tool.
- b. Other standards are required with respect to format of messages and requests for service, as well as handling of torn tape. Two possible standards proposals are given in the appendix, one for handling both upper and lower case graphics on present equipment which has now only upper case, the other a convention for visible graphics which facilitate handling and identification of torn tape.
- c. Some additional features should be provided in the Teletype terminals. These have been requested and are in process. One important element is provision for turning on the remote receiver for full eight-track operation. The Model 35 has manual buttons for TTR and TTS (Tape to Tape Receive and Send). However, manual operation is unacceptable since the highest volume of return from the center will likely be at off hours, around the clock, when most machine shops will depend on unattended operation. Another possible problem is punching of Mylar tape rather than paper tape for durability. Teletype Corporation says that the Model 35 will handle .003 Mylar quite well, but may not do very well on .001 except with brand new punch pins.
- d. If APT III is to become a service language for PCP, some modifications may be required in the existing entrance fee structure and regulations. The role of data centers must be re-examined.
- e. An extensive training program in APT III must be planned. Perhaps the universities will suffice, for they have done an excellent job in the companion language of FORTRAN, but it does not seem that this will cover the present users adequately to forestall job loss and the resultant furor over automation. I suggest that special institutes may have to be formed

for the training of current users of machine tools. This is in direct analogy to the situation in typographical unions with the advent of typesetting by computer. Whose responsibility should this be?

- f. Finally, we need some standard terminology in this field. The APT Task Group of ASA X3.2 should make this one of its first tasks for completion. Here are a few definitions in the general area of PCP provided by Miss Mandalay Grems.

Definitions:

Communication.

The process of conveying information from one point, person or equipment to another.

Communication Link.

The physical means of connecting one location to another for the purpose of transmitting data.

Composition.

In printing, the setting up of type.

Data Transmission.

The process of moving data from one location to another.

Documentation.

The process of collecting, organizing, storing, citing and dispensing of documents or the information recorded in documents.

Timeshare.

To interleave the use of a device for two or more purposes.

Appendix

A. Upper/Lower Case Representation via TWX

In the ASCII code, as represented on current Models 33 and 35, the alphabet is represented by 1 1 1 0 x x x x for upper case. The lower case is scheduled to be 1 1 1 1 s s s s. Obviously the lower case can be fabricated on tape by overpunching the upper case with 1 0 0 1 0 0 0 0, which happens to be the space or blank. To indicate a capitalized word to the computer, punch the letter on the keyboard, backspace the

tape one position (which does not backspace the printing head position) and space. This overpunches the letter representation on tape. The hard copy looks like this:

T HESE W ORDS A RE C APITALIZED

The computer will now invert the representations for the two cases, since what is indicated on the TWX in capitals is really meant to be lower case for text purposes. If by chance the operator should forget to backspace for overpunching it will usually be detectable by the computer program, as the only ambiguous cases occur with initial letter either A or I.

B. Visible Graphics for Identifying Paper Tape

Most commercial Model 33 and 35 Teletype units do not have a facility for imprinting the paper tape with legible text for identification. Thus such identification must be made online by printing the tape.

There is a rather simple way to provide visual identification off-line by punching the graphics of the characters in a 5X7 hole pattern at the leading end of the tape, preferably followed by a string of deletes to separate this identification from the rest of the tape. Inasmuch as Track 8 is always punched (at the present time) there is a difficulty in getting horizontal text to be recognized easily. Furthermore, the available single-key hole patterns do not lend themselves to forming horizontal graphics. On the other hand, vertical text happens to fall out easily. The alphabet and digits are formable by the patterns shown in Table 3.

The character @ (Tracks 7 and 8) is used as the space between graphics, as shown, to provide visual continuity without disruption. The letter O is distinguished from the digit 0 by a dot in the center of the letter.

Aesthetic improvement in the patterns is certainly possible, but it should be noted that the presence of the feed track may throw standard 5X7 patterns out of balance. The other characters of the teletype (ASCII) set are formable in the same manner, but they do not seem necessary to the stated purpose of identification.

This technique may have some advantage in quickly determining the leading end of the tape when not rolled. It may be particularly use-

ful in returning information in tape form from a central computer used as a source of public computing power. It is a trivial matter for the computer to set up this type of output.

A	D J Q Q + Q Q	M	Q U Q Q Q Q	Y	Q J D D D D D
B	+ Q Q + Q Q +	N	Q Y Q U S Q Q	Z	+ B D @ @ H +
C	N Q P P P Q N	O	N Q Q U Q Q N	0	N Q Q Q Q Q N
D	+ Q Q Q Q Q +	P	+ Q Q + P P P	1	D L D D D D D
E	+ P P \ P P +	Q	N Q Q Q U R M	2	N Q A B D H +
F	+ P P \ P P P	R	+ Q Q + T R Q	3	N Q A F A Q N
G	N Q P W Q Q N	S	N Q P N A Q N	4	B F B J + B B
H	Q Q Q - Q Q Q	T	+ D D D D D D	5	+ P + A A Q N
I	N D D D D D N	U	Q Q Q Q Q Q N	6	N Q P + Q Q N
J	A A A A A Q N	V	Q Q Q Q Q J D	7	+ A B D @ H P
K	Q R T X T R Q	W	Q Q Q Q U Q	8	N Q Q N Q Q N
L	P P P P P P +	X	Q Q J D J Q Q	9	N Q Q O A Q N

Table 3

A *priority authorization* allows a job to be processed as soon as possible rather than in turn on a first-in, first-out basis. Certain functions may not be performed without a *modification authorization*. Both authorizations require management approval.

How is ATP used? Debugging a program becomes quite a bit more complex when a program is part of a system that is also being debugged. Many things have to happen at just the right time for just one test run to be executed. The more complex the system, the less chance the individual programmer has of successfully executing a test, let alone getting the right answer.

There are four major steps in the cycle. Once the coding is done, the programmer has to change his module by reassembling it, rebuilding the system with the new module in it, executing the test run, and then checking his results.

A management report is run at the conclusion of every SAVE system function. This report provides information relative to every module in the system. What is in the system? Who is responsible for this module? How much usage and updating has the module received? When was the last time it was used, updated, and stored? How big is the module? What was the initial record and byte count, and what is the current record and byte count? With intelligent use of this report, management can control the frequency of updating and the amount of utilization and growth any module in the system is experiencing. Today, we are studying a project classified as advanced technology. It could result in combining the major features and advantages of the Advanced Terminal System, our Text Writing System and our Automated Test Plan System. This then may well be the possible future system for automating the control of development, distribution, and maintenance of programming systems within the IBM Corporation.

SOFTWARE SYSTEMS CUSTOMIZED BY COMPUTER

R. W. BEMER

*Compagnie Bull General Electric
Paris, France*

Automated production of computer hardware is an accepted practice. Complex tooling, numerically controlled tools, wire-wrap machines, and design automation are employed. Yet until now few manufacturers have given equal effort or consideration to automating software manufacture.

The automobile industry is always fruitful of analogy to computers, so we may say that software is now in the "Black Ford" stage and fabricated by even cruder methods. One would not wish to buy a \$12,000 Ford made by hand to less quality. We must therefore consider software a product and build it by automated methods, because:

1. Customizing has been shown to expand the market.
2. Otherwise software management is going to be very embarrassed to quote 24 to 30 month production cycles, when hardware people can produce a new computer in two months without even a prototype!

The only visible solution is for the manufacturer to utilize the most powerful computer in his (or another, if more advantageous) line as a tool to assist programmers in controlled production and maintenance of software systems for all machines. All functions of software production, documentation, distribution, training, and improvement for all computers should be performed with the aid of an Automated Software Production (ASP) system.

The benefits of such a system should be:

1. Control of production to predicted schedules for predicted costs.
2. At least an order of magnitude increase in reliability and freedom from malfunction.
3. A manyfold reduction in the costs of producing such standard products as FORTRAN and COBOL.
4. Documentation which always matches the current system.
5. Standards of usage across product lines.
6. Systems customized for each user (who prefers to pay for and get only what he needs), with ability to incorporate his own software units and special requirements without interferences or malfunction.
7. Diversion of former wasted effort to further enhancement of software offerings, particularly to generalized applications and corresponding reduction in customer programming required.

To summarize the need, there is almost complete duality between hardware and software in the cycle of research, planning, design, production, delivery, maintenance, support, documentation, obsolescence, and even specially engineered products.

The ASP system provides these functions:

1. Operations *upon* the ASP system.
2. Operations *with* the ASP system.
3. Operations *upon* the user's system.

OPERATIONS

The ASP system provides these functions:

- SLIDE I..... 1. Operations upon the ASP system.
- a. Updating the roster.
 - b. Updating the test library.
 - c. Changes to the ASP system itself, including modifications of its units and logical organization.
2. Operations with the ASP system.
- SLIDE II..... a. Producing provisional systems for temporary programming usage and testing.
- SLIDE III..... b. Producing modifiers to update customer's systems and documentation for distribution.
- c. Producing original manuals and updatings,
 - d. Producing the field report summary and statistics (such as customer batting average).
 - e. Producing records of these processes for the manufacturer.
- SLIDE IV..... 3. Operations upon the user's system.

THE ROSTER

In two sections for each machine type:

General Data

1. Permissible software units supplied without charge.
2. Table of software units keyed to documentation units.

For Each Customer

1. User's name, address and representative.
2. Branch office name, address and representative.
3. Contact pattern between user, branch and programming.
4. Machine type, serial, installation date, on-rent date.
5. Hardware configuration, operational dates of units.
6. Channel assignments, other determinations of logical options.
7. Field change orders affecting software and whether installed or not.
8. Software options for:
 - a. Required units.
 - b. Characteristics of their storage.
 - c. Characteristics of their usage.
 - d. Maximum store allotted for processing and usage.
 - e. Hardware restrictions affecting software operation, such as reserved elements or lockouts.
 - f. Delivery form of software unit (symbolic, relocatable, absolute, FORTRAN, etc.).
 - g. Special software supplementing or replacing standard units, by whom supplied, data descriptions and linkages.

9. Number of last system delivered. Updating pattern and requested frequency (6 month maximum interval for archivage limitation).
 - a. Every system.
 - b. Every nth system.
 - c. Upon specific request.
 - d. First new system after elapsed time interval.
 - e. Only on change to specified software units.
 - f. Combinations of these.
10. Requirements for backup system on another machine.
11. Special commitments by sales or programming personnel.
12. List of customer's field reports by number.

Note: As one user may have multiple machines, this file may be structured with either trailer records or complete duplicates. If the latter, a complete cross-correlation will be necessary.

THE TEST LIBRARY

In four sections:

Roster Consistency

Checks consistency of entries, particularly that hardware or software configurations requested are permissible. If not, that they are either rejected or assessed a special charge.

Program Acceptance Filter

Checks acceptability of any proposed change to a programming system with respect to:

1. Documentation and adequate annotation.
2. Data description.
3. Position of entry or replacement (since a trail must be formed to be able to reconstruct any previous system from the present one).
4. Topological consistency (is anything left useless or destroyed erroneously when needed later?).
5. Adherence to standards (calling sequences, legitimacy of identifiers, operation names and operation pairs).

Quality Tests

These are semi-machine-independent, of types:

1. Logical, such as will the system always return to executive control from any branching? Is the system prevented from doing all that it should not do?

2. Mechanical, such as does FORTRAN handle the expression $B + B + B + \dots + B$ when there are 512 occurrences of B? Included here are generators to create a great variety of source statements to test that processor tables and other elements will handle them correctly. Also included are International (ISO) and country standard test programs. Other programs should be compiled and run, verifying predetermined test answers. These are printed only if they differ, with identification.
3. Operational, such as do all error conditions have an operator message? Simulate the totality and find out.

Field Report Tests

A separate group for each machine, being the total accumulation of reports to date. Each provisional system is required to run all successfully. Thus a mistake corrected on System 6 cannot be reintroduced without warning on System 9, for few things make the customer angrier. Each test is identified by user number for possible deletion if the user is no longer.

THE SIMULATORS

Normally used only to produce original software for a new machine without access to prototype or production model. May also be used for any period of scarcity, such as early testing by customers.

THE GENERALIZED ASSEMBLER

Of the type first developed by CSC and Programmatic, capable of assembling for any object machine provided with an assembly language of this family. Input includes the formats of the source statements and object instructions, together with the transformation rules. It will be necessary to test to prove the identity to the actual assembler as run on the specific machine.



Don McNamara

203 382-4773
Dial Comm: 8*223-4773

3/10/89

Bob,

Congratulations!

Your vision has
made a permanent
impact on our profession

Don

Corporate Information Technology
General Electric Company
1285 Boston Avenue, Bridgeport, CT 06601-2385



The Software Factory: A Historical Interpretation

Michael A. Cusumano, Massachusetts Institute of Technology

Many people associate software factories with Japan. However, one survey shows that Japanese and US software facilities are more similar than not.

A major characteristic of software development in Japan has been the use of the term "software factory" to label development facilities or formal approaches to programming. Factories in other industries have generally mass-produced products including large-scale centralized operations, standardized and de-skilled job tasks, standardized controls, specialized but low-skill workers, divisions of labor, mechanization and automation, and interchangeable parts.

Seeking to benefit from an industrial revolution of its own, engineers and companies in the software industry began using the term "factory" in the 1960s when considering more efficient software-development approaches. This label became especially popular in Japan during the mid-1970s and 1980s.

Software differs from conventional, "hard" products made from interchangeable components and constructed through a sequential assembly process; it is

primarily an iterative process of design, coding, testing, and redesign. There are only a few industry-wide standards for product features, tools, or project-management techniques. So is there sufficient base to apply factory concepts to software?

Organizational specialists have also warned that unstandardized, complex technologies like software in still-evolving markets are not suitable for highly structured, factory-like processes. Instead, they require ad hoc responses from highly skilled workers — as you would find in a craft-oriented job shop. So is it even appropriate to apply factory concepts to software?

Furthermore, one firm's adoption of the term "factory" does not mean that its practices are necessarily different from firms that do not use the label. For example, a 1983 survey¹ found about 200 enterprises in the US alone that had more than 1,000 software personnel in centralized locations, with many using stan-

standardized designs and reusable code components, centralized tool development, formal testing and quality-assurance departments, productivity-measurement and productivity-improvement efforts, and research on the development process. Although there was no mention in this study of how systematic US firms were in their management practices, centralization, standardization, reusability, and control are factory-like concepts, even though no US firm used the term "factory" to describe its facility. So what is a software factory?

What degree of integration or standardization among tools, methods, controls, and skills might distinguish a factory mode of operation from simply a large group of people working more or less independently in the same facility?

Furthermore, what can we learn about software engineering from Japanese software factories? Or were the Japanese merely labeling their facilities differently?

This article tries to provide some answers.

Definitions

The first public proposals for the adoption of factory-type methods and organizations for software appeared in the late 1960s as outgrowths of comparisons of programming with engineering and manufacturing practices. Perhaps the earliest proponent, R.W. Bemer of General Electric, made many proposals that culminated in a 1968 paper² suggesting that General Electric develop a software factory to reduce variability in programmer productivity through standardized tools, a computer-based interface, and a historical database useful for financial and management controls. (General Electric's exit from the computer business in 1970 ended the company's commitment to commercial hardware and software production.)

Bemer's paper gave the first working definition of what might constitute a software factory: "A software factory should be a programming environment residing upon and controlled by a computer. Program construction, checkout, and usage should be done entirely within this environment and by using the tools contained in the environment. ... A factory ...

has measures and controls for productivity and quality. Financial records are kept for costing and scheduling. Thus, management is able to estimate from previous data. ... Among the tools to be available in the environment should be compilers for machine-independent languages; simulators, instrumentation devices, and test cases as accumulated; documentation tools — automatic flow-charters, text editors, [and] indexers; accounting function devices; linkage and interface verifiers; [and] code filters (and many others)."

While Bemer focused on standardized tools and controls, M.D. McIlroy of AT&T emphasized another factory-like concept: systematic reusability of code when constructing new programs. In an address at a 1968 NATO science conference on software engineering,³ McIlroy argued that the division of programs into modules

It seemed too difficult to create modules that would be efficient and reliable for all types of systems and that did not constrain the user.

offered opportunities for mass-production methods. He then used the term "factory" in the context of facilities dedicated to producing parameterized families of software parts or routines that would serve as building blocks for tailored programs reusable across different computers.

Reaction to McIlroy's ideas was mixed: It seemed too difficult to create modules that would be efficient and reliable for all types of systems and that did not constrain the user. Software was also heavily dependent on the specific characteristics of hardware. Nor did anyone know how to catalog program modules so they could be easily found and reused. Nonetheless, by the late 1960s, the term "factory" had arrived in software engineering and was being associated with computer-aided tools, management-control systems, modularization, and reusability.

The comments of Bemer and McIlroy show that US firms had been grappling with large-scale programming efforts since the late 1950s. Like General Electric and AT&T, IBM made many discoveries about how and how not to manage software when it deployed a thousand or more programmers during the mid-1960s to develop the operating systems for the System 360 family of mainframes. IBM's facilities for basic software tried to standardize methods for all phases of development during the late 1960s and 1970s and introduced a variety of tools and management controls. In this sense, IBM and other large-scale software producers in the US and Europe were at least contemplating factory-like procedures and organizational structures by the late 1960s.

The establishment of IBM's Santa Teresa Laboratory in California in the mid-1970s also brought together 2,000 programmers in one site and reflected IBM's continuing attempts to structure its basic software-development operations. Thus, you might argue that the term "factory" implicitly referred to good software-engineering practices applied systematically, at least within a facility, although US companies did not place much emphasis on McIlroy's factory concept of reusability.

First software factory

The first company in the world to adopt the term "factory" (actually, its Japanese equivalent, "kojo," which translates as either "factory" or "works") to label a software facility was Hitachi, which founded the Hitachi Software Works in 1969. A history of independent factories for each major product area prompted executives in Hitachi's computer division to create a separate facility for software when this became a major activity.

Hitachi managers set two goals for their factory: (1) productivity and reliability improvement through process standardization and control and (2) the transformation of software from an unstructured service to a product with a guaranteed level of quality. This was necessary to offset both a severe shortage of skilled programmers in Japan and the many complaints from customers about bugs in Hitachi's software (most of which, along with the hardware,

Hitachi was importing from RCA until 1970).

The fact that all Hitachi factories had to adopt corporate accounting and administrative standards forced software managers to analyze the development process in great detail and experiment with a series of work standards and controls. The independence of Hitachi factories within the corporation also gave factory managers considerable authority over technology development. Managers concentrated initially on determining factory standards for productivity and costs in all phases of development, based on standardized databases for project management and quality control.

Hitachi then standardized design around structured-programming techniques in the early 1970s and reinforced these standards with training programs for new employees and managers. This reflected an attempt to standardize and improve average skills rather than specify every procedure to be performed in each project and in each development phase.

After some success in process control, Hitachi invested extensively in automated tools for project management, design support, testing, program generation, and reuse support.

At the same time, however, Hitachi managers underestimated how difficult implementing factory concepts such as reusability and process standardization would be. For example, their attempt in the early 1970s to introduce a component-control system for reusability failed, as did efforts to introduce one standardized process for both basic software and custom applications software. The need to separate methods and tools for different software types led to a separate division for basic software and applications at Hitachi Software Works and then to the 1985 establishment of a second software factory dedicated to applications.

Second software factory

While Hitachi managers struggled with the meaning and limitations of factory practices, one US leader in the custom software field, System Development Corp. (formerly a part of the Rand Corp. and now a Unisys division), established the world's second software factory in 1975-

1976. SDC had been separated from Rand in the 1950s to develop Sage, the Semi-automatic Ground Environment missile-control system, for the US Defense Dept. It later took on other real-time programming tasks as a special government-sponsored corporation, but it finally went public in 1970. Top management then had to control software costs and so launched a factory-oriented R&D effort in 1972 to tackle problems^{4,5} its programmers continually faced:

- Lack of disciplined and standardized approaches to the development process.
- Lack of an effective way to visualize and control the production process, including ways to measure before a project was completed how well code implemented a design.
- Difficulty in accurately specifying performance requirements before detailed

Project managers did not like giving up control to a centralized facility, and, surprisingly, top management did not require that they use the software factory.

design and coding, including recurring disagreements on the meaning of certain requirements and changes demanded by the customer.

- Lack of standardized design, management, and verification tools, making it necessary to reinvent them from project to project.

- Little capability to reuse components, even though many application areas used similar logic and managers believed that extensive use of off-the-shelf software modules would significantly shorten software-development time.

More so than at Hitachi, SDC engineers constructed a detailed plan for a factory process and organization with three elements:

- an integrated set of tools (program library, project databases, on-line interfaces between tools and databases, and automated support systems for verification and

documentation).

- standardized procedures and management policies for program design and implementation, and
- a matrix organization separating high-level system design (at customer sites) for program development (at the software factory).

The first site to use the factory system was a facility of about 200 programmers in Santa Monica, Calif. SDC even copyrighted the name "The Software Factory."

Scheduling and budget accuracy improved dramatically for 10 projects that went through the factory, but management ended the effort in 1978 for two reasons.

First, programmers found it was extremely difficult without portable computer languages to reuse code and tools from one project for different applications and for different computers.

Second, and more important, the tradition in SDC had been for project managers to create programming groups that would work at individual customer sites, in a mobile, job-shop production mode. They did not like giving up control of development efforts to a centralized facility, and, surprisingly, top management did not require that they use the software factory. This led to a decline in the flow of work into the facility as project managers built their own teams on customer sites. Ultimately, by removing programmers from the factory, it faded out of existence when the last project in the factory ended.

In retrospect, SDC managers tried to impose a factory infrastructure of standardized tools and methods and reusability goals on a range of projects that were too different and better suited to job-shop production. The state of software technology at the time made it difficult to transport tools and code across different machines. Furthermore, architects of the factory failed to solve the organizational problems that resulted from separating design from product construction. These problems, which were mainly resistance from project managers, prevented a steady work flow into the facility.

Nonetheless, while SDC abandoned its factory effort, the company continued to use many of the factory procedures and

Table 1.
Japanese software-factory organizations.

Year established	Company	Facility/project	1988 products*	1987-88 employees
1969	Hitachi	Hitachi Software Works	Basic systems	1,500
1976	NEC	Software Strategy Project Fuchu Mita Mita Abiko Tamagawa	Basic systems Industrial real-time control General business applications Telecommunications Telecommunications	2,500 2,500 1,250 1,500 1,500
1977	Toshiba	Fuchu Software Factory	Industrial real-time control	2,300
1979	Fujitsu	Kamata Software Factory	General business applications	1,500
1983	Fujitsu	Numazu Software Division	Basic systems	3,000
1985	Hitachi	Omori Software Works	General business applications	1,500
1985	NTT	Software Development Division	Telecommunications	400
1987	Mitsubishi	Computer Factory	Basic systems, general business	700

*Basic systems include operating systems, database-management systems, and language utilities.

some of the tools. The SDC model also influenced the software standards later developed both by the US Defense Dept. and by factory efforts already under way in Japan.

Software-factory boom

Following SDC's announcement of its software factory in 1975, Japan's leading hardware and software manufacturers in addition to Hitachi — NEC, Toshiba, and Fujitsu — launched their own factory efforts during 1976-1977 (see Table 1).

Toshiba created what is perhaps the most structured factory — although it also had the most focused product lines — using a centralized software facility to develop real-time process-control software for industrial applications. Similarities in this type of software from project to project let Toshiba build semicustomized programs by combining reusable designs and code with newly written modules rather than write all software from scratch.

The Toshiba system,^{6,7} built around its Software-Engineering Workbench, uses a version of the Unix environment, a full complement of tools for design support, reusable-module identification, code generation, documentation and maintenance,

testing, and project management. An important feature of the Toshiba approach was the design of new program modules (generally limited to 50 lines) for reusability, the requirement that programmers deposit a certain number of reusable modules in a library each month, and the factoring of reuse objectives into project schedules.

While Toshiba was building its factory, NEC began its efforts to rationalize soft-

ware production in its computer division's main factory, which developed operating systems and other basic software. Management then set up a research laboratory in 1980 to direct the development of tools, procedures, and design methods for various types of software, and it organized a quality-assurance effort to standardize management practices throughout the corporation. Because product divisions did not always accept technology from the laboratory, NEC has let divisions modify tools or add their own design or production-control systems. The result was a mixture of standardized processes and tools with variations that correspond to product areas.⁸

Fujitsu established a basic software division in its hardware factory in the mid-1970s and then launched a separate software factory for applications programming in the late 1970s. Its initial goal was to convert Hitachi and IBM programs to run on Fujitsu machines. Fujitsu then placed considerable emphasis on developing automated design-support tools for the production of business-applications programs.⁹

Fujitsu, Hitachi, and NEC have avidly promoted the diffusion of factory-type practices by transferring their tools and processes to other in-house facilities, as well as to subsidiaries, subcontractors, and hardware customers.

More recently, Mitsubishi and Nippon Telephone and Telegraph have begun

Japan's leading hardware and software manufacturers in addition to Hitachi launched their own factory efforts during 1976-1977.

factory-like efforts. Like other Japanese firms, Mitsubishi is emphasizing reuse through modifying (reengineering) code or designs to make them applicable to more than one project. NTT also established a centralized software division that serves largely as a design factory, handing off many specifications to suppliers for implementation.

Perhaps the most significant new factory-like effort is the national Sigma project started in 1985 by Japan's Ministry of International Trade and Industry. The project has more than 130 corporate members, including several non-Japanese firms. The five-year, \$200-million project is a bold attempt to create a national communications network connecting 10,000 sites, standardized workstations using a Unix-based environment, educational programs for members, and a library of reusable program parts and software-development tools. If it succeeds, the project will significantly raise the level of support tools and knowledge available to smaller Japanese software manufacturers.

Comparing approaches

To a large degree, establishing factory programs signals a commitment to long-term, integrated efforts — above the level of individual projects — to structure, standardize, and support development along the lines suggested by software-engineering literature since the late 1960s.

Some firms have needed to do this more than others — for example, if they were short of experienced personnel and competed in markets where other firms might offer comparable or better software more quickly and cheaply. In Japan during the 1970s, there was a special urgency to improve levels of productivity, quality, and process control to offset shortages of skilled programmers and good software packages and to accommodate rapid rises in demand, especially for complex customized applications programs and lengthy basic software for their new hardware systems introduced to compete with the IBM System 370.

Although each had different products or emphases, in pursuing more engineering-like and manufacturing-like practices, Hitachi, Toshiba, NEC, and Fujitsu largely followed IBM and other US leaders

Survey of management emphases

Descriptions and objectives for System Development Corp.'s software factory provided a basis for my drawing up eight criteria cutting across inputs standardization (emphasis on reuse of software code), tool standardization and integration, and process standardization and control. I identified major software producers in Japan and the US (and one in Canada) through literature surveys and lists of software producers; further investigation led to the identification of senior managers either responsible for overall software-engineering management or with responsibilities over several projects and with sufficient experience to present an overview of practices for an entire facility or division. I then surveyed those managers.

Managers who agreed to participate in the survey were asked to rank their emphasis and impression of general policy at their facilities on a scale of 0 to 4 and to comment on each answer. Optional questions also requested performance measures such as actual rates of reused code in a recent sample year. The sample was limited to facilities or departments making operating systems for mainframes or minicomputers (systems software) and a variety of applications programs. All the Japanese firms contacted filled out the survey; about three quarters of the US firms contacted completed the survey.

Factor analysis indicated that the eight questions dealt with two distinct factors: The inputs and tools questions combined as one factor while process questions remained a separate factor. The two factors explained 79.6 percent of the variance in the survey answers; the inputs factor alone accounted for 58.8 percent of the variance. I then tested differences in the average Japanese and North American scores and whether product type or country of origin of the facility were significantly correlated with the factor scores.

Table A summarizes the average Japanese and North American responses to the inputs and tools questions and to the process questions. The results, although exploratory, support the hypothesis that there is a spectrum among managers of how they view software development. Despite potential views of software development as largely a craft, art, or job-shop type of operation, some managers at facilities making similar types of products clearly placed more emphasis on control and standardization of inputs (reusable modules of code) and basic tools and processes. Analysis of variance tests confirmed that product types had no significant effect on where managers scored on the dimensions surveyed.

A second hypothesis you might derive from the differences between the Japanese and US software markets is that there are national differences, with the Japanese significantly emphasizing reusability more than US firms. Confidentially reported Japanese reuse rates were also significantly higher than North American rates (34.8 percent versus 15.4 percent) across all product types, although this and other performance data are tentative because of possible differences in counting at different firms.

Table A.
Comparison of average
Japanese and North American survey scores.

Dimension	Japanese score σ^*	North American score σ^*	Average score σ^*
Inputs**	8.7 2.1	6.0 2.7	7.3 2.4
Tools/process	14.4 2.9	15.0 3.7	14.7 3.3

*standard deviation; **significance level is 0.01

and thus pursued remarkably similar paths toward a more structured approach to software development. Their efforts began with decisions to create centralized organizations and management-control systems for specific product families, standardize around methods and tools tailored to these products, and then provide partially automated support for development and project management. After establishing this foundation, they began making refinements as well as more automated and flexible tools capable of handling different languages or tasks (see Figure 1).

So what, if anything, distinguishes Ja-

panese software factories from other large-scale facilities? The box above shows an initial attempt to explore this question through a survey I conducted. The survey indicated that more than half the participating 51 facilities or divisions making software for mainframes or minicomputers (25 Japanese, 25 from the US, and one from Canada) could be characterized as flexible factories in the sense that managers strongly emphasized code reuse as well as standardization and control over tools and processes — at least as reflected in a few questions — and made unique basic software or customized applications software. Other facilities appeared more

Phase 1: Basic organization and management structure (mid-1960s to early 1970s)

- Factory objectives established
- Product focus determined
- Process data collection and analysis begun

Phase 2: Technology tailoring and standardization (early 1970s to early 1980s)

- Control systems and objectives established
- Standard methods adopted for design, coding, testing, documentation, and maintenance
- On-line development through terminals
- Employee training program to standardize skills
- Program libraries introduced
- Integrated methodology and tool development begun

Phase 3: Process mechanization and support (late 1970s to present)

- Introduction of tools supporting project control
- Introduction of tools to generate code, test cases, and documentation
- Integration of tools with on-line databases and engineering workbenches begun

Phase 4: Process refinement and extension

- Revision of standards
- Introduction of new methods and tools
- Establishment of quality-control and quality-circle programs
- Transfer of methods and tools to subsidiaries, subcontractors, and hardware customers

Phase 5: Flexible automation

- Increase in capabilities of existing tools
- Introduction of reuse-support tools
- Introduction of design-automation tools
- Introduction of requirements-analysis tools
- Further integration of tools through engineering workbenches

Figure 1. Software-factory process evolution.

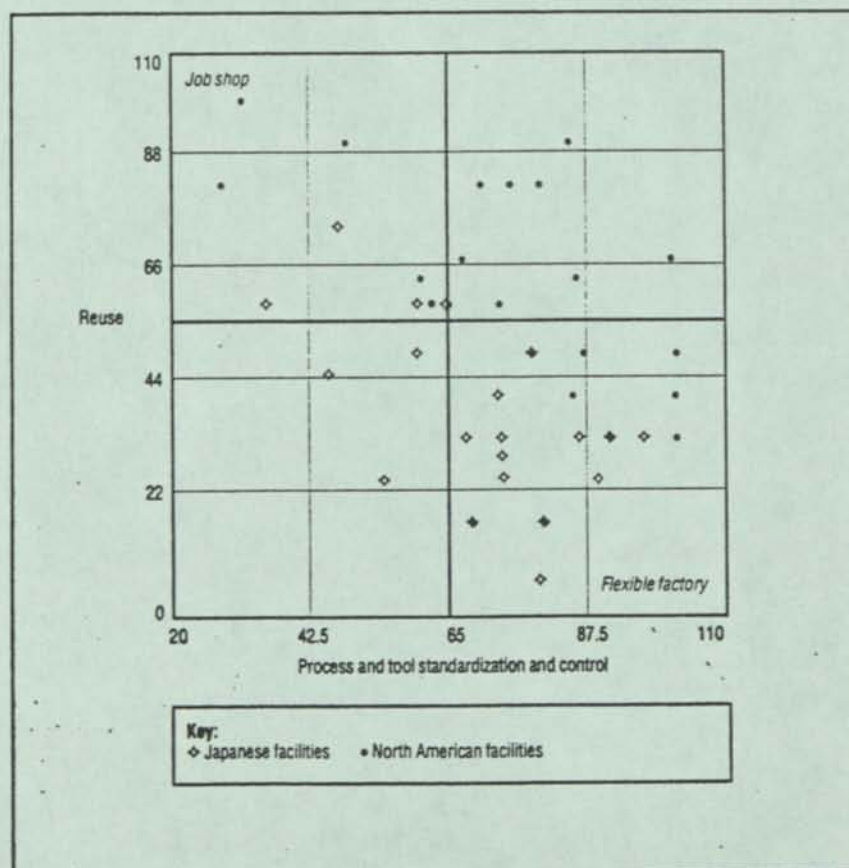


Figure 2. Emphasis on reuse versus emphasis on tool and process standardization for Japanese and North American firms.

like job shops, since managers placed little or no emphasis on standardization, control, or reusability (see Figure 2).

While the survey was exploratory and not a comprehensive analysis of tools, quality practices, or other aspects of software development, it suggested three points:

- Software producers in Japan and North America both stress some factory-like practices and fall into a spectrum of implementations, with some managers placing more emphasis on aspects of standardization, control, and reusability than others.

- This spectrum cuts across different product types, defined as basic software, general business applications, real-time control systems, industrial operating systems, and telecommunications software.

- Although there are not significant differences in how managers in large Japanese and US firms responded to the questions on standardization and control over tools and process elements, the Japanese placed significantly higher emphasis on a basic factory-like concept: reusability.

The Japanese concern with reuse reflects the strategies of software factories like those at Toshiba, Fujitsu, and Hitachi, which try to build customized applications programs by combining existing code (or designs) with new code. Reuse seems highly appropriate as a response to a shortage of programmers, high demand for customized applications, and low sales (or low availability) of mass-market packages — characteristics that describe the Japanese software market.

According to data from the US Commerce Dept. and Japan's Information Service Industry Association, in 1986 as much as 94 percent of the software sold in Japan (excluding systems software, which was generally included with hardware) was fully or partially customized or designed for integrated hardware and software systems (see Table 2). By contrast, nearly 60 percent of US domestic software sold in 1986 were mass-market packages. More rapid growth in package sales from 1987 to 1988 brought the percentage of customized software in Japan down to about 85 percent as the sales of both small computers and packages have increased, but

many Japanese buyers continued to prefer customized software for their larger computers.

This emphasis on customized software placed tremendous demands on Japanese software producers faced with a shortage of skilled programmers and a backlog of orders, especially in earlier years when they were still refining their production systems. In other product areas, Japanese firms were trying to leverage their design skills across different projects, such as re-using compiler designs across different operating systems.

Making case studies of Japanese software factories and comparing them to other facilities confirmed the survey's findings that tools and methods are essentially similar in Japan and the U.S. although the large Japanese firms appeared especially concerned with process improvement and left product innovation largely to their U.S. counterparts. Japanese managers seemed intensely dedicated to standardizing good practices, gradually improving tools and techniques, and strategically integrating their efforts with rigorous employee training.

They also developed product and marketing strategies to segment users and tailor process technology to particular products and customers. For example, Japanese software factories generally made products similar to systems they had made in the past. For totally new projects, managers channeled work to less-structured subsidiaries, software manufacturers, or special projects outside the factories.

What constituted a software factory was difficult to measure precisely, although structured facilities clearly seemed different from job-shop approaches in their degree of focus on relatively routine designs, standardized methods and tools, standardized training, and controls and integration of these elements above the level of the individual project.

Process Improvement

However you label the approaches of Japanese firms, they appeared effective in improving productivity, quality, and process control over what they had done earlier. They also appear to compare favorably to current U.S. levels, although

Table 2.
Software market data (1985-86).

	Japan	U.S.
Total market revenues	\$5 billion	\$19 billion
Total package software	6 percent	65 percent
Total custom software	94 percent	35 percent
Microcomputer software's share of total market	10 percent	40 percent
Annual increase in demand for software	25 percent	25 percent
Annual growth in supply of programmers	13 percent	4 percent
Typical wait for customized program	26 months	40 months
Computer manufacturers as suppliers of systems software	70 percent	45 percent
applications software	15 percent	5 percent

such performance comparisons are difficult to make.

Hitachi doubled productivity between just 1969 and 1970 after introducing factory standards. Productivity stagnated during the 1970s as Hitachi worked on refining its tools and methods, but it began rising rapidly again in the late 1970s and 1980s, especially after the introduction of reuse-support and automated program-

However you label the approaches of Japanese firms, they appeared effective in improving productivity, quality, and process control over what they had done earlier.

ming tools for business-applications development. Hitachi also reduced the amount of late projects from 72 percent in 1970 to 7 percent in 1974 and has since averaged about 12-percent late projects a year. Bugs per machine in the field dropped from an index of 100 in 1978 to 13 in 1984.

Toshiba improved productivity from an equivalent of 1,390 assembly lines of source code per month (about 460 in Fortran) in 1976 to 3,100 per month (about 1,033 in Fortran) in 1985 — including reused code totaling 48 percent of delivered

lines. At the same time, quality improved from seven to 20 faults per thousand lines of source code at the end of final test to about two or three faults per thousand. Residual faults left after testing averaged between 0.05 and 0.2 per thousand lines.

NEC saw productivity improvements of 26 percent to 91 percent by using a set of standardized procedures, tools, and reusable patterns for applications software — while at the same time reducing bugs about one third. Fujitsu cut bugs for all outstanding operating-systems code (newly delivered code plus maintained systems) from 0.19 per thousand lines in 1977 to 0.01 per thousand in 1985. Productivity in lines of code per month also improved by two thirds between 1975 and 1983.

The management skills and tool support perfected at software factories also seemed to help develop enormously complex software with high productivity and reliability. Toshiba, for example, has been a world leader in real-time automated process-control software for electric and nuclear power plants since the late 1970s.

Data collected in 1988 by myself and Kent Wallgreen, a graduate student, also indicates that the Japanese consistently showed higher output in lines of code over time compared to more experienced U.S. programmers. The difference seemed to be that Japanese firms reused larger amounts of code. They also seemed to finish their products quickly rather than optimize designs by reducing lines of code.

These practices tended to make modules longer, thus making productivity appear higher when you use lines-of-code measures.

In the quality area, however, the Japanese appeared to be outstanding: Their projects generally averaged as much as 50-percent fewer bugs per thousand lines of delivered code and required less maintenance time as a percentage of total development time compared to US projects. This is especially impressive because maintenance in Japan is usually performed by separate departments whose members were not the original developers of the software, so they must maintain code they are unfamiliar with.

As the demand for and complexity of software continue to increase, the ability to compensate for shortages of skilled programmers through standardization of skills, introduction of good

procedures and tools, refinement of project-management techniques, and advanced technology for reuse support and automated programming will become even more important to the software industry. These are factory-like concepts and technologies, whether firms adopt the factory label or not, and Japanese factories appear to be among the world leaders in developing them.

Japanese firms have had a weakness: their focus on process improvement rather than product innovation or package development. Both for this reason and because the demand for software in Japan still exceeds the ability of domestic firms to supply it, Japanese companies have not exported many programs or bid very frequently overseas for contract software. Many people have thus concluded that Japanese firms, because they have little export presence, are weak in software. However, this does not seem to be true — at

least among the large firms.

Furthermore, Japanese managers appear to be confident that they have mastered many of the basic problems in software development, such as quality control and project management, and are now ready to tackle the systematic improvement of product functionality and ease of use. In fact, recent survey data comparing the responses of Japanese customers to Japanese- and US-made software indicates that the large Japanese firms have achieved parity or superiority in custom applications, although they still trail in basic systems software.^{10,11}

If Japanese achievements in other industries or in computer hardware are any indication, these new emphases in Japanese software factories are likely to lead to even more functional, low-cost, and reliable software in the future that may compete more directly with US products, especially customized applications software. ❖

References

1. C. Jones, *Programming Productivity*, McGraw-Hill, New York, 1986, p. 243.
2. R.W. Bemer, "Position Papers for Panel Discussion: The Economics of Program Production," in *Information Processing 68*, North-Holland, Amsterdam, 1969, pp. 1626-1627.
3. M.D. McIlroy, "Mass-Produced Software Components," in *Software Engineering: Reports on a Conference Sponsored by the NATO Science Committee*, P. Naur and B. Randell, eds., Scientific Affairs Div., NATO, Brussels, 1969, pp. 151-155.
4. H. Bratman and T. Court, "The Software Factory," *Computer*, May 1975, pp. 28-37.
5. "Elements of the Software Factory: Standards, Procedures, and Tools," in *Software-Engineering Techniques*, Infotech Int'l, Berkshire, England, 1977, pp. 117-143.
6. Y. Matsumoto, "SWB System: A Software Factory," in *Software-Engineering Environments*, H. Hunke, ed., North-Holland, Amsterdam, 1981.
7. Y. Matsumoto, "A Software Factory: An Overall Approach to Software Production," in *Software Reusability*, P. Freeman, ed., CS Press, Los Alamitos, Calif., 1987.
8. K. Fujino, "Software Development for Computers and Communications at NEC," *Computer*, Nov. 1984, pp. 57-62.
9. N. Murakami et al., "SDEM and SDSS: An Overall Approach to the Improvement of the Software-Development Process," in *Software-Engineering Environments*, H. Hunke, ed., North-Holland, Amsterdam, 1981.
10. "Dai-2 Kai SE Sabisu Kanren Chosa" (Second Survey on Systems-Engineering Service), *Nikkei Computer*, March 14, 1988, pp. 58-86 (in Japanese).
11. "Dai-5 Kai Oyo-Konpyuta-Yuza Sensasu" (Fifth Census on General-Purpose-Computer Users), *Nikkei Computer*, Sept. 26, 1988, pp. 66-99 (in Japanese).



Michael A. Cusumano is an assistant professor at the Massachusetts Institute of Technology's Sloan School of Management, where he teaches courses on corporate strategy and technology management. He is also interested in hardware technology transfer and product development in the computer industry, product development and manufacturing in consumer electronics, and the transfer of Japanese production techniques to other countries. He is the author of a book forthcoming from Oxford University Press called *The Software Factory*. Cusumano received a PhD from Harvard University in Japanese studies. He is fluent in Japanese and has lived and worked in Japan for more than five years, including three years as a Fulbright fellow and researcher on economics and social science at the University of Tokyo.

Address questions to the author at Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139.

FIELD REPORT

MAIN STREET



ONLINE AUCTION. Boston's public TV station conducted a special kind of auction last month, using only an online service, not TV and telephones, to sell computer-related products to the highest bidders. WGBH's history of auctions, and its proximity to high-tech companies, had brought the station many donations of specialized or high-priced items that would be difficult to explain or merchandise "traditionally." So the online auction was conceived, recalled auction manager Edye Baker, seen reviewing the system with David Solomont, president of Business and Professional Software and a WGBH online auction advisor. Donated products included software packages, training seminars, books, modems and other peripherals, plus supplies and accessories.

TRENDS

"Factories" For Software?

Japanese methods boost quality, productivity

How does .01 defects per program package per installation per year sound for your in-house systems? Could you tolerate a productivity level of 800-1,000 lines of code per staff month, if your in-house systems were nearly defect-free?

That's the promise brought by the "software factory" approach which a handful of Japanese companies are now using successfully, according to Donald M. McNamara, program manager in General Electric's Corporate Information Technology group in Bridgeport, Conn.

McNamara told an audience in the distinguished lecturer series at the Wang Institute of Graduate Studies (Tyngsboro, Mass.) that bringing a factory mentality to software development has worked in Japan; but his audience pointed out that cultural differences could hinder its adoption in the U.S.

The factory concept means using

development tools such as 4GLs and other automated development tools, re-using code, and implementing quality assurance at every step.

"Design code for re-use, and register it. Re-use is a secret to productivity," McNamara said.

Other issues to be faced include a willingness by users to: commit to a requirements definition; accept the risk that standard methodologies impede innovation; work long hours during implementation; and overcome the "not invented here" syndrome.

But the payoffs can be significant, he said.

In two examples, Japanese companies spent the equivalent of millions of dollars to establish their software factories, and because of this they had top-management attention. They were able to replicate their systems and avoid controversy and duplication. The developers were able to remain focused throughout the process, he reported.

NETWORKING

Mac LANs: Birth Trauma

User training is required: Scully

Apple Chairman John Scully comes right to the point: "This is no longer a box industry," he said. "It's system-software driven."

"Apple had to learn this the hard way. We never shipped the file server we announced two years ago; it was harder than we thought to connect computers and third-party products."

As a result, he candidly admitted users will need specialized training.

The company's newly-announced file server, Appleshare, was "difficult to give birth to, but will be rewarding in the future," Scully said. "It's built on the Mac's foundation—a consistent set of development rules and a rich environment for applications."

"The Mac was controversial in 1984," he continued. "Functionality was limited but it was built on next-generation technology. Now we have a firm foundation, while the DOS world faces issues of change as they move into their next generation, i.e. the 80386 CPU."

Scully said that networking Macintoshes requires dedicating one Mac as a server because the software connections are complicated enough to demand specialized training.

"Not all of our dealers and resellers are authorized to handle communications products," he said. "They need to have a systems engineer who takes our training courses and exams. The user needs to designate a network administrator who will receive our training—though the course of study for that can be sold through a reseller."

To potential Appleshare users, he cautioned, "Don't think about getting into our communications products without taking the course: it's complicated. But you don't have to be a technical person to become a network administrator."

—Hal Glatzer

Economics of programming production

by: Robert W. Bemer

1. INTRODUCTION

A software consultant recently advised a large computer manufacturer that each programmer should write 8 instructions per hour. I believe that he also stipulated that the sequence should be valid. The astute programmer will immediately write a generator for the computer to produce valid sequences of 10 000 instructions per hour and depart for the Riviera. I should prefer getting a product of maximum utility.

Thinking of what this might have meant in the remote computer past (such as 1956), I recall doing the PRINT I system for the IBM 705 at a completed cost of \$17 per instruction. Since it was a compact semi-interpretive system of some 1200 instructions in all, the cost did not seem at all out of line. Yet you may be assured that the programmers were not paid at the rate of (8×17) \$136 per hour less machine time! The SAGE system of 1957 cost \$50 per average instruction, so we must assume that they did not write 8 per hour.

Taking this to a more modern absurdity, consider a typical Fortran processor of perhaps 35 000 instructions. According to the rule this would require 4375 hours. At U. S. A. rates of \$10 per hour per programmer, it would be delightful to get such a system for a mere \$43 750. More likely it will be \$437 500 until we are better able to mechanize such production. Actually it is almost possible in the particular case of Fortran [1], but I mean to specify the everyday situation.

All this leads to the point that various applications of computers have various complexities, and cannot be accomplished to rigorous and invariant standards. Not that I do not believe in standards, but in the case of programming I must say that I simply do not know what a programmer should produce. That is the business of his manager. What I do say is that when the methods outlined in this paper are followed, any type of programming can be produced at a very great saving over the usual methods of today.

2. DOCUMENTATION

2.1. *Functional specifications*

Programmers are prone to build without plan. This is the most expensive method for any type of architecture. Even starting with a complete flowchart is nevertheless building without a plan, for they are not equivalent! One may have a perfect flowchart for the wrong process.

The members of each level in the programming hierarchy, from product planner to detail coder, should be obliged to write down in a formal manner the following information according to their responsibility:

What is the purpose of the program unit?

What are the inputs and their forms?

What are the outputs and their forms?

What processes are applied to the inputs to yield outputs?

What is the inventory of tools (usable store, utility routines, other program units, executive controls) available?

What are the constraints of time and interaction with other program units?

What are the operational design goals and characteristics?

What are the characteristics of interface with other program units?

These functional specifications should be completely settled before any flowcharting, programming or coding is attempted. They should be matched against similar specifications for other units to detect either conflict, duplication, or imbalance in the system. Duplication in particular is a major cost item. Proper functional specifications allow programmer A to find out that he has a similar need to programmer B, such that they can share a subroutine.

2.2. *Operating characteristics*

Some of the greatest losses in computer efficiency occur when unbuffered decisions must be taken by human operators. Therefore the preliminary design should always state the operating characteristics of the program units as embedded in the entire system. Among the items to be specified are:

Error conditions and messages.

Restart conditions and necessary actions.

Complete alternatives to all possible decisions.

2.3. *Descriptions and manuals*

Every program unit must have some descriptive material associated to provide a permanent record of the characteristics which affect the user of the program. Functional specifications do not necessarily have to be made available to the user, as they may belong properly to "technical documentation" which does not have this requirement. This material may consist of

but a single sheet of paper, or it may be a complete manual in the case of large and complicated programs. In either case it is advisable to insist that a rough draft be made before any flowcharting or coding is attempted. Obviously certain characteristics cannot be known until the program runs, but it is preferable to indicate this in the original version by a note such as "method unknown - could be ..., or ..., or ...". As the decisions are made, the draft should be updated. This acts as a constant reminder and prevents overlooking of design needs.

When possible, it is preferable for the documentation of minor programs to be integral with the program in the form of annotation. This should lead to prevention of program changes without corresponding changes to documentation. However, formal methods of control should be exercised to ensure that the narrative is still consistent as changed.

2.4. Listings

Listings are the backbone of documentation. However, in early production it is difficult to properly balance use of handwritten correction and notes, on the one hand, with amendment by complete reprocessing on the other. Patches are to be avoided unless they may be accomplished by the most foolproof and mechanical methods.

One must keep good records at any stage of development. In 1963 a large software house did a major system by keeping changes solely as a superimposition of various tape systems, without benefit of updated listings. In the instability of early development there were a few times when the current version was destroyed, necessitating a reconstruction period of up to 2 weeks to recover the current system. At their rates, each occurrence lost up to \$80000! Doing it all by punched cards and overnight off-line listings would have been cheaper. It is always better to spend extra effort to keep the cleanest possible record, so that each iteration may be taken as a complete restarting point.

There is a simple compromise. Allow a little extra space on each listing page by not filling it completely. Give the top line of each page an artificial identifier if it does not already have one, all of which are kept in a list associated with page number. Since most modern programming systems produce coding relocatable by hardware or software, actual store assignments are somewhat irrelevant. Thus it will often be necessary to make a new listing page selectively only where programming changes occur.

3. STANDARDS

3.1. Terminology

A major way of lowering programming costs, often ignored, is to im-

Obviously the last item could often be in multiple, and could therefore be compacted in a tabular form.

4.2. *Flowcharting and logic equations*

Programs should be carefully designed, by whatever means. Flowcharts enjoy a certain popularity for clarity. However, they are usually not so necessary when programming in a language like Cobol. Logic equations have the capability of being formally manipulated for minimization or seeing that all negations are accounted for.

4.3. *Modularity*

Always build a program of any size in discrete modules, with known inputs and outputs, together with the interior process. These should be so independent that they may be linked together in almost any order, just like railroad cars. This might require 3% more instructions overall, but it is worth it in costs of maintenance and diagnosis.

Every program unit should be created in three forms for testing:

- (a) As a self-contained unit, complete with synthetic input and output, created perhaps by a generator.
- (b) In a form suitable for usage within its own major program.
- (c) In a form suitable for use within the overall system.

Often the extra instructions required for (a) and (b) may be removed mechanically for the final stage.

5. PRODUCTION CONTROL

Due to the invisibility of programs, normal control methods are ineffective. Mechanized control and feedback is even more important than the precise organization of supervision. The steps are:

5.1. *Estimation and budget*

Software units of the minimum size feasible for individual control are defined, named and given identifying numbers. Planning provides a working description of function. Supervisors estimate the total elapsed time and cost for man- and machine-hours. This is the primary input to the budget. In the case of large concerns with many programmers at different locations, precise definition of a programming unit to be fabricated allows for competitive bidding among these groups, with corresponding expectancy of cost reduction.

5.2. *Labour distribution*

Supervisors distribute the total elapsed time by benchmarks (functional

prove the communication between contributing programmers. Since many programs now have international utility, it is advisable to adopt terminology from the only internationally agreed effort, the IFIP/ICC Vocabulary of Information Processing [2]. This work is structured by concept, and is worthy of careful study prior to usage for looking up individual terms for reinforcement. Missing or newly developed concepts should be brought to the attention of the IFIP/ICC committee.

3.2. *Other standards*

When costs are a consideration, it is foolish to program without a minimum of standards. There should be an active standards unit in every production programming group, policing compliance with national and international standards as available [3, 4, 5]. In addition there must be internal, local standards on such items as:

- Consistency of appearance and documentation.

- Calling sequences.

- Description of programming units with respect to algorithm of solution, restrictions, degenerate cases, range, valid classes of data, test cases, etc. It is advisable to adopt a widely tested method to be able to interchange and use the programs of others for economy [6].

4. DESIGN

4.1. *Checklists*

Because of the nature of the work, a programmer usually desires to invent something. However, given a variety of previous wheel designs it is likely that he will spend this effort on something not so often re-invented. This is the purpose of the checklist [7]. It recognizes that most programming problems are of a highly recurrent nature. It also recognizes that total recall of all contingencies or ways of doing things is unlikely for most programmers, just as the doctor does not always remember the totality of symptoms without aid. For example, it is trivial for the programmer to check off or complete such items as:

- The source code for this assembly system may come from (punched cards, paper tape, magnetic tape, OCR *,...)

- If the computer stops with (give here a combination of conditions), the operator should (...).

- Data named (...) are (always/often/never) (numeric/alphabetic/(other)) and require (...) positions on a (punched card/paper tape) in the format (...) and position defined by (...).

* Optical character reading.

specifications, flowcharts, implicit quality test, coding, checkout in vacuo, checkout in processor, checkout in system, documentation, explicit quality test, release). Labour distribution reports are developed by means of time cards. These are correlated to the estimates. The individual programmer periodically estimates the percentage of completion of each unit. If the system is run on a computer, it is possible to flag estimated overruns in hours and delivery times, inconsistencies in reporting precedence dangers on PERT schedules, etc.

5.3. *Correction and adjustment*

Supervisors add revised benchmark estimates to project charts, which show initial estimates, last revised estimates and actual completions. In danger areas, management may rebalance the staff, redesign, etc. The eventual users of the programs are notified of revised dates so they may modify plans, check contractual commitments. As these are official company records, detected falsifiers may be discharged, as merited discipline is usually effective in reducing costs. The supervisors may be recalibrated as optimists or pessimists, but more often they will automatically adjust their estimating as a byproduct of the system. Data present themselves for practical standards of production, in those areas where it is feasible to have such standards.

6. DIAGNOSTIC METHODS

6.1. *General*

Computer operation has become more complex with each year of usage. Not only are translators for the Fortran, Algol and Cobol languages used widely, but even the assembly languages have become more complex. All of these now run under *executive* systems likely to become more intricate than they are. Under such conditions, the programmer is likely to be at a loss to find out whether a malfunction is due to:

- a hardware malfunction;
- a malfunction in the programming system he is using;
- an operating mistake;
- data errors, such as unexpected type, outside of expected range, physical errors in preparation or reading, etc.;
- his mistake, such as a misunderstanding or disregard for the rules of syntax, grammar, construction, file layout, system configuration, flow process for solution, etc.

The hardware field engineer is subject to the same confusion. However, there are certain ways of discovering the class of the malfunction and directing the evidence to the proper authority for correction. The program-

mer should not be too surprised if, after following the methods outlined here, this turns out to be himself in most cases.

When using a programming system, remember that there is probably no single person that understands the entire system and its individual components well enough to diagnose 100% of the troubles. This means that most diagnosis must be done by *cause* and *effect*, rather than tracing through the operation. The "black box" simile must be appreciated and used. One must put certain inputs into the box, observing the form of the outputs. One then varies the inputs and observes the corresponding changes (if any) to the form of the outputs. By careful design of the inputs and their variation it is possible to deduce which internal element of the black box must be at fault.

This means that the programmer must adopt the scientific method of "design of experiment". The object is to get as much information as possible during each run (or experiment) and to make as few runs as possible. Thus many items of information should be obtained from each run, but the variations must not interfere with each other to the extent of obscuring information, and each bit of information should lead to the next set of modifications by reducing the possibilities.

Before the user can call upon outside help, it is his responsibility to clearly demonstrate the malfunction. Further, he should provide the *minimum* segment of the program which exhibits the malfunction. Thus isolation is the first process to undertake.

It is much cheaper to be prepared for a malfunction than not. A good rule to adopt is that "the program is wrong when first ready for testing". The unusual ("degenerate" to the mathematician) case occurs when the program is correct just prior to production runs. Cases are known where the average number of times to compile or assemble a program for test was in excess of fifty before it operated satisfactorily. This is too expensive and delays production to an intolerable point [8].

6.2. Practical methods

6.2.1. Multiple service per run

There are few things as shameful as seeing a programmer run a program to blowup point, take a full dump of the store and get off the machine. This is expensive in machine time and slows his productivity. Observe the following program structure:

Read initial values of parameters;

* List values as read;

Compute A, B and C;

* Read correct values for A, B and C according to the initial values given. Call them a, b and c.

* Compute $A - a$, $B - b$ and $C - c$. If all zero, print "A, B and C OK" and jump to "Next step". If not, print:

A = ...	a = ...
B = ...	b = ...
C = ...	c = ... , and

* Compute $A = a$, $B = b$ and $C = c$.

Next step.

The steps marked with an asterisk should normally be removed only when testing is complete and correct. This can often be done automatically during final compilation by a switch mechanism. Do not remove in stages, as correct sections may be again incorrect upon changes.

It takes little effort to adopt this plan, particularly if called by a Fortran procedure or PROC (super-macro) [9]. It ensures that the next program segment can be checked independently in the same run. Good practice dictates that the programmer divide into at least ten such parts per run!

6.2.2. *Controlled data*

Allowing complete freedom of data characteristics during original testing can introduce too many complexities to see clearly what is going wrong. Select certain values for inputs and run them through the algorithm to determine the expected results for selected combinations. Make the selection according to these criteria:

- For numeric parameters, take values at the end points of expected allowable range.
- For non-numeric parameters, take typical or singular cases that display all expected characteristics.
- In either case, vary for minimum and maximum field length.
- Select "bad" data with specific characteristics such that they should not work in the program.
- To check moves, do the inverse and compare to itself, like a matrix reinversion. Build this in and remove when correct.

Test to determine that all valid data yield correct answers and that the bad data always yield error conditions and messages. Subtract check answers from actual and blank zeros before printing.

6.2.3. *Live data*

Live data should be used only after obtaining correct results with controlled data. In case of malfunction, check that the live data:

- conform to data characteristics which the documentation shows to affect program action, and match format rules;
- come from the proper physical input unit;
- do not contain invalid characters, singly or in combination.

Check the answer range. Overflow and underflow truncated can give unrecognizable answers.

6.2.4. *Desk checking*

Machine time is still expensive enough to warrant considerable desk checking. I say this despite any claims in this area for online man-machine interaction with timesharing. The programmer should take the steps that follow.

- (a) Check conformity to rules, such as those for justification.
- (b) See that enough restart points exist for long programs.
- (c) Compare actual program logic for match with intended logic as given by a flowchart or equation.
- (d) Examine live input for peculiar characteristics which could cause erroneous branching, such as bad data, blank records, etc.
- (e) Inspect the list of identifiers produced and assigned by the processor, looking for conflicts, insufficient definition, completeness and spelling.
- (f) Check permissible spellings of reserved words, allowable usage of spacing, hyphens and commas, and juxtaposition of illegal word or operation pairs.

Obviously much of this should be detected by a well-designed processor with complete error message facility, but this is not always so.

6.2.5. *Branching*

When the decision structure of a program is at all complex, always plan a path flow in the testing. This may be as simple as printing the value and name of the element tested, printing a suitable indicator for branch or no branch. It is good to print, in the extreme righthand columns if convenient, a code or label of the first instruction in the branch to identify the branch selected. Print this during execution of the branch sequence, not when the decision is made!

When the proper branch is not taken for some reason, invert both the test and the branch destinations. For example, the following program segments are identical in function:

If A = B, go to P G : P	If A ≠ B, go to G go to P G : P
--	---

If they work differently, it is obvious that the mistake lies in obtaining the form of A and B.

6.2.6. *Operating*

The goal of operating the program in the test environment should be to develop the simplest and smallest program segment which exhibits the malfunction, regardless of whether the eventual cause is shown to be the

responsibility of hardware, the software system or the user. To this end take the following steps.

- (a) Reduce the program in size and complexity.
 - (b) Isolate suspected sections of coding and equip them to run individually, but in groups one after the other. Test to see if the malfunction has disappeared. If not, add original elements until it reappears.
 - (c) Simplify the section of coding. Replace arithmetic statements by simple statements like $A = B$. Simplify variable names. Put complex flow in line.
 - (d) Check all diagnostic messages for clues.
 - (e) Check to see if dual or complement types of instructions also cause the malfunction, or simply an expected wrong answer.
 - (f) Make several physical copies of the malfunctioning section. Vary in several ways, adjoin copies and run together for efficiency.
 - (g) Reprogram for alternate methods of achieving the same result. This is often the simplest way to overcome blindness to the cause of malfunction.
 - (h) In difficult cases, change values of only one variable at a time for controlled experiment.
 - (i) Make full use of manufacturer-supplied tools such as de-flowcharting, dynamic testing routines, utilities, etc.
- Additional treatment may be found in [10].

6.2.7. *Quality control*

The best way to avoid malfunctions is to build software with quality controls applied during manufacture. All original programming, changes and additions to programs are done preferably in a computer-controlled environment [11, 12]. Such environments should be in general use by computer manufacturers by 1967, and should be available to users then for similar usage.

REFERENCES

1. Digitek Fortran (Advertisement), *Datamation* 1964 Aug, 35-38.
2. A. R. Wilde et al., *IFIP/ICC Vocabulary of information processing* (North-Holland Publishing Co., Amsterdam, 1965).
3. International Standards Organization, Technical Committee 97, *Computers and information processing* (Scope, Geneva, 1961).
4. ISO/TC97/WG-G (Secr-29)62, Second draft proposal, *Flowchart symbols for information processing* (ASA, New York, 1964).
5. ISO/TC97/SC2 (Secr-37)130 F/E, Fourth draft proposal, *6 and 7 bit coded character sets for information processing interchange* (AFNOR, Paris 2^e, 1965).
6. M. Grems, Proposal for an ACM-JUG computer applications digest (Minutes ACM Council, 1965).

7. R. W. Bemer, A checklist of intelligence for programming systems, Communications ACM 2 (1959) 8-13.
8. M. E. Senko, A control system for logical block diagnosis with data loading, Communications ACM 3 (1960) 236-240.
9. UNIVAC, General Manual, Sleuth II for 1107, UP-3670, 1963.
10. UNIVAC, P.I.E. Bulletin UP-3910.5, 1964.
11. R. W. Bemer, Software systems customized by computer, Proceedings IFIP Congress 65, Vol. II.
12. W. R. Crowley, A possible future system for automating control of the development, distribution and maintenance of programming systems, Proceedings IFIP Congress 65, Vol. II.

PROGRESS IN HARDWARE AND SOFTWARE

R.W.Bemer, Bull General Electric, Paris

Presented to the MC/E International Data Processing Conference
Milano, 1966 March 9-11

ABSTRACT

Increasing complexity in computer systems, especially in the real-time area, heightens the interaction between software and hardware. Thus the previously separate functions of field engineering maintenance and software support tend to depend more on each other and even merge, particularly in the diagnosis of malfunction. It is advisable to use a computer network for fabrication, distribution and maintenance of software, particularly when (as in the case of Bull General Electric) there are multi-country sources for hardware and software system components, and a subsidiary structure for sales and service.

In the production system now being considered, one or more central computers are connected by communications links to systems programmers, sales offices and customer installations. Motivations for such a system are:

- Optimization of European programming talent
- Rapid and efficient reporting and correction of malfunctions
- Common Access files for sales control, queries and contracts
- Direct distribution of programs and documentation
- Controlled automated production of software
- Upgrading support personnel for better customer service

Optimization of European Programming Talent

Europeans are less flexible for physical movement than are Americans, due to language differences, working papers, equivalents of social security, taxes, etc. It is therefore expensive and difficult to set up a central programming site and staff it multi-nationally. Some examples exist of both failure and less than complete success.

Software by its basic nature requires centralized control for fabrication, largely because of its relative invisibility. However, it is not especially critical where this control is located with respect to the other functions of the computer manufacturer, subject to sufficient liaison. How then to make best use of the considerable programming talent in Europe?

One characteristic of time-sharing unexpected by most people is the effectiveness of human-to-human communication between terminals. A classic example is that of the programmer demonstrating the JOVIAL language to another. Each time he wrote a statement, a passive observer at another terminal corrected it. Finally he was forced to type a message to the person interfering, asking if he could please make his own mistakes, as he wanted to demonstrate the detection features in the system.

Such systems are certainly interactive to the extent required for intercommunication in joint remote production of software systems. Furthermore, software tools may be constructed for the central computer which monitor and control this production process, filtering out unacceptable, incompatible and non-standard inputs in a more effective way than possible for the human programming supervisor.

Consider the rule that statements in assembler language can not be introduced into the software unless at least ten characters of comment are associated. The programmer who does not comply with this desirable practice will not have a high production count on his monthly ranking, which will give his human supervisor a much better indication of his worth. Thus, given a small facility in English, and adequate communication linkages throughout Europe, there could be an alternate method of multinational software construction.

Preparation for this type of work is presently under way in London, where we have assembled a multinational group to build a major software system. In many ways the methods remain the same as before. However, the group is headed by an American team of absolutely top ranking which is enjoined to:

- 1) Teach all techniques of good software production while the project is in operation.
- 2) Demonstrate all the techniques of good software project management used, with particular attention to costs and meeting of schedules.
- 3) Make available to every participant the schedules and proposed costs, so that they may simulate project management as in a case study.
- 4) Indoctrinate in the usage of the new General Electric documentation models, that maximum information will be made available to prospective users at the earlier moment, in a necessarily complete and standardized format.

It is of interest to know that in the planning stage of this project, which was carried out by three of the American team and

four from France, over 700 pages of specifications were produced in six weeks. The Phoenix organization called this the most remarkable effort of this type that had been seen, thus further reinforcing my views that European programmers are every bit as capable as American, given equivalent facilities and supervision. Thus we hope to make a graceful transition in our subsidiaries, upgrading our personnel to full software construction capability, while still proceeding cautiously enough for adjustment when goals are missed. The time scale is not felt to be critical, as there are no expectations that European communications networks will all be of sufficiently high and uniform quality for this type of operation for at least three more years.

Rapid and Efficient Reporting and Correction of Malfunctions

A remote terminal is assumed to be a necessary adjunct to every computer installation, existing either on site or at the sales office. It is also assumed that qualified systems programmers are engaged in remote production of new software while servicing current software and assisting customers in its proper usage.

When a field engineer is confronted with a malfunction not immediately correctable on site, he uses the remote unit to transmit details to the software maintenance and control computer. As a part of the automated software production system, it is easy to determine that the originator of the software unit (e.g., SIGMA - a structural engineering program) is still online to the system, last calling in from Portofino (he is based at Olivetti General Electric in Milano). Details are therefore relayed to his terminal.

If the system on which the malfunction occurred is of the same type as the central production computer, direct action may be taken. If not, the programmer must arrange for testing on a comparable system, which may sometimes be the reporting system itself. The cause may be hardware, software or user malfunction. If a software change must be made, the necessary instructions are entered into the central system, together with any documentation changes caused by the repair. They are distributed to the originating system and to all other systems allowing open access to that software unit at any time. Other users will wait until scheduled periodic reissue of the system with accumulated changes.

Common Access Files for Sales Control, Queries and Contracts

A system for automated software production requires complete files on equipment installed and on order; information must be complete on configurations (both hardware and software), software production and delivery schedules, authorized software units, associated manuals, etc. A sales office connected by terminal unit to this central system may therefore verify accuracy and availability of the items included in his sales proposal. Normally such basic information will also be distributed to the field sales force in the form of a price list or sales data book. Such information is subject to change, however, and errors or loss are possible. This easy access to the single official information set is a desirable precaution.

Of particular importance is the requirement not to contract for unrealistic delivery schedules, failure of which could lead to contract abridgement. Neither customer nor manufacturer wants to be deluded in this respect. The central system provides the actual expected schedules, together with a margin for contractual acceptability. If schedules in a proposed contract fall beyond this limit, then the salesman is authorized to conclude the order without further approval in this area. Thus the system can serve as an order acceptance filter and update the on-order file. It should not be too difficult to transmit this file or changes to the factory to use as the official basis for factory order and production schedules.

Direct connection to sales offices should facilitate usage of

computer programs to evaluate computer performance. Bull General Electric uses BULLRAC (SCERT) to a great extent. While such programs may not give perfect simulation in every respect, they do yield inequalities which are more useful for evaluation than cycle times, tape density and other foolishness. Here the customer is served again, for he will be protected from unwise investments in computer systems which will not perform as expected.

The last benefit to be mentioned is that of optimization. The central system maintains all schedules; the contractual requirements are grouped and associated with the various hardware and software units. This information may be used to manipulate production priorities to give optimum customer satisfaction.

Direct Distribution of Programs and Documentation

While it is true that large volumes of programming systems and related documentation would be distributed by mail, as for the original issuance of a system, minor changes and additions would be distributed via the remote terminals. This could take the form of a change program on paper tape for presentation to the modification filter of the software system, and would contain the necessary keys to unlock the system so that it could modify itself while still running under its own control. This applies even to the executive control unit itself, although it may seem a little like sitting in one's own lap.

Distribution of basic and applications software can be made selectively according to a predetermined interest profile, or the customer could request programs of interest based upon abstracts sent to the terminals. In some instances the customer might be allowed to request design specifications in order to comment prior to construction of the program. Timely advice and news can also be made available as soon as confirmed.

For highly customized software, the documentation required by the customer is not only a subset of the total, but may vary internally for each installation. For this reason, it is preferable for documentation to be keyed to the software units and closely associated with them in the distribution process.

Upgrading Support Personnel for Better Customer Service

The real costs of programming and analysis are usually much less when performed by experienced and qualified persons. Their salaries will be higher, to be sure, but not in the proportion of the greater efficiency effected by the superior machine processing which they enable. It is therefore desirable to have highly qualified personnel engaging in customer assistance, if only on a part-time advisory basis.

However, it is difficult to ask such people to do this type of work if no other challenging assignments are at hand. One of the advantageous features of the remote production method is that field reporting and maintenance may be done by experts who reside physically near to customer installations. Since this is only a part-time assignment, the balance of their time may be spent in creative software production and even research. Furthermore, working directly with customers yields a closer association with the mix of actual problems to be solved. This experience is very valuable in the design and production of new systems, for the needs and interests of customers can be represented in both the design and quality assurance testing phases.

Misc Notes

Salesman may query for other similar applications on other customer computers. Can search x for sales points.

Check if configuration is highly recommended or not. Ranking for various software units.

Files are OPEN, CLOSED, or ADDITIVE to MGMT, Programmers, Customers, Field Engrs, sales reprs, customer support technicians

Possible terms:	Custodian	Guardian
	Dispatcher	Parent
	Director	Mother
	Duenna	Overseer
	Governor	Tender (a ala subma-ine)
		Service

ECONOMICS OF PROGRAMMING PRODUCTION

more time and less money

by ROBERT W. BEMER

A presentation at the Symposium on the Economics of ADP, held last October in Rome, Italy, this article appears in the proceedings of the symposium, which is available from the North-Holland Publishing Co., Amsterdam, The Netherlands.

A software consultant recently advised a large computer manufacturer that each programmer should write eight instructions per hour. I believe that he also stipulated that the sequence should be valid. The astute programmer will immediately write a generator for the computer to produce valid sequences of 10,000 instructions per hour and depart for the Riviera. I should prefer getting a product of maximum utility.

Thinking of what this might have meant in the remote computer past (such as 1956), I recall doing the PRINT I system for the IBM 705 at a completed cost of \$17 per instruction. Since it was a compact semi-interpretive system of some 1200 instructions in all, the cost did not seem at all out of line. Yet you may be assured that the programmers were not paid at the rate of (8 x 17) \$136 per hour less machine time! The SAGE system of 1957 cost \$50 per average instruction, so we must assume that they did not write eight per hour.

Taking this to a more modern absurdity, consider a typical FORTRAN processor of perhaps 35,000 instructions. According to the rule this would require 4,375 hours. At U.S.A. rates of \$10 per hour per programmer, it would be delightful to get such a system for a mere \$43,750. More likely it will be \$437,500 until we are better able to mechanize such production. Actually it is almost possible in the particular case of FORTRAN¹, but I mean to specify the everyday situation.

All this leads to the point that various applications of computers have various complexities, and cannot be accomplished under rigorous and invariant standards. Not that I do not believe in standards, but in the case of programming I must say that I simply do not know what a programmer should produce. That is the business of his manager. What I do say is that when the methods outlined in this paper are followed, any type of programming can be produced at a very great saving over the usual methods of today.

documentation

Functional Specifications. Programmers are prone to build without plan. This is the most expensive method for any type of architecture. Even starting with a complete flowchart is nevertheless building without a plan, for they are not equivalent—one may have a perfect flowchart for the wrong process.

The members of each level in the programming hierarchy, from product planner to detail coder, should be obliged to write down in a formal manner the following information according to their responsibility:

1. What is the purpose of the program unit?
2. What are the inputs and their forms?
3. What are the outputs and their forms?
4. What processes are applied to the inputs to yield outputs?
5. What is the inventory of tools (usable store, utility routines, other program units, executive controls) available?
6. What are the constraints of time and interaction with other program units?
7. What are the operational design goals and characteristics?
8. What are the characteristics of interface with other program units?



Mr. Bemer, who has been working with computers for 17 years, is a software consultant to the General Electric Co. He has also been associated with Univac as director of system programming, and with IBM as director of programming standards. He was a member of the ACM Council for six years, represented AFIPS on the IFIP/ICC vocabulary committee, and was a major participant in the development of the ISO/ASCII code.

¹ Digitek Fortran (Advertisement), *Datamation* Aug., 1964, pp. 35-38.

These functional specifications should be completely settled before any flowcharting, programming or coding is attempted. They should be matched against similar specifications for other units to detect either conflict, duplication, or imbalance in the system. Duplication in particular is a major cost item. Proper functional specifications allow Programmer A to find out that he has a similar need to Programmer B, such that they can share a subroutine.

Operating Characteristics. Some of the greatest losses in computer efficiency occur when unbuffered decisions must be taken by human operators. Therefore the preliminary design should always state the operating characteristics of the program units as embedded in the entire system. Among the items to be specified are:

1. Error conditions and messages.
2. Restart conditions and necessary actions.
3. Complete alternatives to all possible decisions.

Descriptions and Manuals Every program unit must have some descriptive material associated to provide a permanent record of the characteristics which affect the user of the program. Functional specifications do not necessarily have to be made available to the user, as they may belong properly to "technical documentation" which does not have this requirement. This material may consist of but a single sheet of paper, or it may be a complete manual in the case of large and complicated programs. In either case it is advisable to insist that a rough draft be made before any flowcharting or coding is attempted. Obviously certain characteristics cannot be known until the program runs, but it is preferable to indicate this in the original version by a note such as "method unknown—could be, or, or" As the decisions are made, the draft should be updated. This acts as a constant reminder and prevents overlooking of design needs.

When possible, it is preferable for the documentation of minor programs to be integral with the program in the form of annotation. This should lead to prevention of program changes without corresponding changes to documentation. However, formal methods of control should be exercised to ensure that the narrative is still consistent as changed.

standards

Terminology. A major way of lowering programming costs, often ignored, is to better the communication between contributing programmers. Since many programs now have international utility, it is advisable to adopt terminology from the only internationally agreed effort, the IFIP/ICC Vocabulary of Information Processing.² This work is structured by concept, and is worthy of careful study prior to usage for looking up individual terms for reinforcement. Missing or newly developed concepts should be brought to the attention of the IFIP/ICC committee.

Other Standards. When costs are a consideration, it is foolish to program without a minimum of standards. There should be an active standards unit in every production programming group, policing compliance with national and international standards as available.^{3,4,5} In addition there must be internal, local standards on such items as:

1. Consistency of appearance and documentation.
2. Calling sequences.
3. Description of programming units with respect to algorithm of solution, restrictions, degenerate cases, range, valid classes of data, test cases, etc. It is advisable to adopt a widely tested method to be able to interchange and use the programs of others for economy.⁶

Listings. Listings are the backbone of documentation. However, in early production it is difficult to properly

balance use of handwritten correction and notes, on the one hand, with amendment by complete reprocessing on the other. Patches are to be avoided unless they may be accomplished by the most foolproof and mechanical methods.

One must keep good records at any stage of development. In 1963 a large software house did a major system by keeping changes solely as a superimposition of various tape systems, without benefit of updated listing. In the instability of early development there were a few times when the current version was destroyed, necessitating a reconstruction period of up to two weeks to recover the current system. At their rates, each occurrence lost up to \$80,000! Doing it all by punch cards and overnight off-line listings would have been cheaper. It is always better to spend extra effort to keep the cleanest possible record, so that each iteration may be taken as a complete re-starting point.

There is a simple compromise. Allow a little extra space on each listing page by not filling it completely. Give the top line of each page an artificial identifier if it does not already have one, all of which are kept in a list associated with page number. Since most modern programming systems produce coding relocatable by hardware or software, actual store assignments are somewhat irrelevant. Thus it will often be necessary to make a new listing page selectively only where programming changes occur.

design

Checklists. Because of the nature of the work, a programmer usually desires to invent something. However, given a variety of previous wheel designs it is likely that he will spend this effort on something not so often re-invented. This is the purpose of the checklist.⁷ It recognizes that most programming problems are of a highly recurrent nature. It also recognizes that total recall of all contingencies or ways of doing things is unlikely for most programmers, just as the doctor does not always remember the totality of symptoms without aid. For example, it is trivial for the programmer to check off or complete such items as:

1. The source code for this assembly system may come from (punch cards, paper tape, magnetic tape, OCR,)
2. If the computer stops with (give here a combination of conditions), the operator should (.)
3. Data named (. . .) are (always/often/never (numeric/alphabetic/ (other)) and require (. . .) positions on a (punch card/paper tape) in the format (.) and position defined by (.).

Obviously the last item could often be in multiple, and could therefore be compacted in a tabular form.

Flowcharting and Logic Equations. Programs should be carefully designed, by whatever means. Flowcharts enjoy a certain popularity for clarity. However, they are usually not so necessary when programming in a language like COBOL. Logic equations have the capability of

² Wilde, A. R., et al, IFIP/ICC Vocabulary of Information Processing, North-Holland Publishing Co., Amsterdam, 1965.

³ International Standards Organization, Technical Committee 97, Computers and Information Processing, Scope, Geneva, 1961.

⁴ ISO/TC97/WG-G (Secr-29)62, Second Draft Proposal, Flowchart Symbols for Information Processing, ASA, New York, Aug., 1964.

⁵ ISO/TC97/SC2 (Secr-37)130/FE, Fourth Draft Proposal, 6 and 7 bit Coded Character Sets for Information Processing Interchange, AFNOR, Paris 2^e, March, 1965.

⁶ Grems, M., Proposal for an ACM-JUG Computer Applications Digest, Minutes ACM Council, May, 1965.

⁷ Bemser, R. W., A Checklist of Intelligence for Programming Systems, Communications ACM 2, March, 1959, pp. 8-13.

being formally manipulated for minimization or seeing that all negations are accounted for.

Modularity. Always build a program of any size in discrete modules, with known inputs and outputs, together with the interior process. These should be so independent that they may be linked together in almost any order, just like railroad cars. This might require 3% more instructions overall, but it is worth it in costs of maintenance and diagnosis.

Every program unit should be created in three forms for testing:

1. As a self-contained unit, complete with synthetic input output, created perhaps by a generator.
 2. In a form suitable for usage within its own major program.
 3. In a form suitable for use within the overall system.
- Often the extra instructions required for (1) and (2) may be removed mechanically for the final stage.

production control

Due to the invisibility of programs, normal control methods are ineffective. Mechanized control and feedback is even more important than the precise organization of supervision. The steps are:

Estimation and Budget. Software units of the minimum size feasible for individual control are defined, named and given identifying numbers. Planning provides a working description of function. Supervisors estimate the total elapsed time and cost for man- and machine-hours. This is the primary input to the budget. In the case of large concerns with many programmers at different locations, precise definition of a programming unit to be fabricated allows for competitive bidding among these groups, with corresponding expectancy of cost reduction.

Labor Distribution. Supervisors distribute the total elapsed time by benchmarks (functional specifications, flow-charts, implicit quality test, coding, checkout in vacuo, checkout in processor, checkout in system, documentation, explicit quality test, release). Labor distribution reports are developed by means of time cards. These are correlated to the estimates. The individual programmer periodically estimates the percentage of completion of each unit. If the system is run on a computer, it is possible to flag estimated overruns in hours and delivery times, inconsistencies in reporting precedence dangers on PERT schedules, etc.

Correction and Adjustment. Supervisors add revised benchmark estimates to project charts, which show initial estimates, last revised estimates and actual completions. In danger areas, management may rebalance the staff, redesign, etc. The eventual users of the programs are notified of revised dates so they may modify plans, check contractual commitments. As these are official company records, detected falsifiers may be discharged, as merited discipline is usually effective in reducing costs. The supervisors may be recalibrated as optimists or pessimists, but more often they will automatically adjust their estimating as a byproduct of the system. Data presents itself for practical standards of production, in those areas where it is feasible to have such standards.

diagnostic methods

General. Computer operation has become more complex with each year of usage. Part of this is attributable to the wider use of FORTRAN, ALGOL and COBOL languages. But even assembly languages have become more complex, and all of these now run under executive systems likely

to be more intricate than the languages. Under such conditions, the programmer is likely to be at a loss to find out whether a malfunction is due to:

1. A hardware malfunction.
2. A malfunction in the programming system he is using.
3. An operating mistake.
4. Data errors, such as unexpected type, outside of expected range, physical errors in preparation or reading, etc.
5. His mistake, such as a misunderstanding or disregard for the rules of syntax, grammar, construction, file layout, system configuration, flow process for solution, etc.

The hardware field engineer is subject to the same confusion. However, there are certain ways of discovering the class of the malfunction and directing the evidence to the proper authority for correction. The programmer should not be too surprised if, after following the methods outlined here, this turns out to be himself in most cases.

When using a programming system, remember that there is probably no single person that understands the entire system and its individual components well enough to diagnose 100% of the troubles. This means that most diagnosis must be done by *cause* and *effect*, rather than tracing through the operation. The "black box" simile must be appreciated and used. One must put certain inputs into the box, observing the form of the outputs. One then varies the inputs and observes the corresponding changes (if any) to the form of the outputs. By careful design of the inputs and their variation it is possible to deduce which internal element of the black box must be at fault.

This means that the programmer must adopt the scientific method of "design of experiment." The object is to get as much information as possible during each run (or experiment) and to make as few runs as possible. Thus many items of information should be obtained from each run, but the variations must not interfere with each other to the extent of obscuring information, and each bit of information should lead to the next set of modifications by reducing the possibilities.

Before the user can call upon outside help, it is his responsibility to clearly demonstrate the malfunction. Further, he should provide the *minimum* segment of the program which exhibits the malfunction. Thus isolation is the first process to undertake.

It is much cheaper to be prepared for a malfunction than not. A good rule to adopt is that "The program is wrong when first ready for testing." The unusual ("degenerate" to the mathematician) case occurs when the program is correct just prior to production runs. Cases are known where the average number of times to compile or assemble a program for test was in excess of fifty before it operated satisfactorily. This is too expensive and delays production to an intolerable point.⁸

Practical Methods. 1. Multiple Service per Run. There are few things as shameful as seeing a programmer run a program to blowup point, take a full dump of the store and get off the machine. This is expensive in machine time and slows his productivity. Observe the following program structure:

- Read initial values of parameters
 - List values as read
 - Compute A, B and C
 - Read correct values for A, B and C according to the initial values given. Call them a, b and c.
 - Compute $A = a$, $B = b$ and $C = c$.
- If all zero, print "A, B and C OK" and jump to "Next"

⁸ Senko, M. E., *A Control System for Logical Block Diagnosis with Data Loading*, Communications ACM 3 (1960), pp. 236-240.

step"

If not, print

A = a =
 B = b =
 C = c = , and

* Compute A = a, B = b and C = c.

Next step

The steps marked with an asterisk should normally be removed only when testing is complete and correct. This can often be done automatically during final compilation by a switch mechanism. Do not remove in stages, as correct sections may be again incorrect upon changes.

It takes little effort to adopt this plan, particularly if called by a FORTRAN subpositive.⁹ It ensures that the next program segment can be checked independently in the same run. Good practice dictates that the programmer divide into at least ten such parts per run!

2. *Controlled Data.* Allowing complete freedom of data characteristics during original testing can introduce too many complexities to see clearly what is going wrong. Select certain values for inputs and run them through the algorithm to determine the expected results for selected combinations. Make the selection according to:

- For numeric parameters, take values at the end points of expected or allowable range.
- For non-numeric parameters, take typical or singular cases that display all expected characteristics.
- In either case, vary for minimum and maximum field length.
- Select "bad" data with specific characteristics such that they should not work in the program.
- To check moves, do the inverse and compare to itself, like a matrix reinversion. Build this in and remove when correct.

Test to determine that all valid data yields correct answers and that the bad data always yields error conditions and messages. Subtract check answers from actual and blank zeros before printing.

3. *Live Data.* Live data should be used only after obtaining correct results with controlled data. In case of malfunction, check that the live data:

- Conforms to data characteristics which the documentation shows to affect program action, and matches format rules.
- Comes from the proper physical input unit.
- Does not contain invalid characters, singly or in combination.

Check the answer range. Overflow and underflow truncate can give unrecognizable answers.

4. *Desk Checking.* Machine time is still expensive enough to warrant considerable desk checking. I say this despite any claims in this area for on-line man-machine interaction with time-sharing. The programmer should:

- Check conformity to rules, such as those for justification.
- See that enough restart points exist for long programs.
- Compare actual program logic for match with intended logic for match with intended logic as given by a flow chart or equation.
- Examine live input for peculiar characteristics which could cause erroneous branching, such as bad data, blank records, etc.
- Inspect the list of identifiers produced and assigned by the processor, looking for conflicts, insufficient definition, completeness and spelling.
- Check permissible spellings of reserved words, allow-

able usage of spacing, hyphens and commas, and juxtaposition of illegal word or operation pairs.

Obviously much of this should be detected by a well-designed processor with complete error message facility, but this is not always so.

5. *Branching.* When the decision structure of a program is at all complex, always plan a path flow in the testing. This may be as simple as printing the value and name of the element tested, printing a suitable indicator for branch or no branch. It is good to print, in the extreme righthand columns if convenient, a code or label of the first instruction in the branch to identify the branch selected. Print this during execution of the branch sequence, not when the decision is made!

When the proper branch is not taken for some reason, invert both the test and the branch destinations. For example, the following program segments are identical in function:

IF A = B, GO TO P	IF A ≠ B, GO TO G
G	GO TO P
.	G
.	.
P	P

If they work differently, it is obvious that the mistake lies in obtaining the form of A and B.

6. *Operating.* The goal of operating the program in the test environment should be to develop the simplest and smallest program segment which exhibits the malfunction, regardless of whether the eventual cause is shown to be the responsibility of hardware, the software system or the user. To this end:

- Reduce the program in size and complexity.
- Isolate suspected sections of coding and equip them to run individually, but in groups one after the other. Test to see if the malfunction has disappeared. If not, add original elements until it reappears.
- Simplify the section of coding. Replace arithmetic statements by simple statements like A=B. Simplify variable names. Put complex flow in line.
- Check all diagnostic messages for clues.
- Check to see if dual or complement types of instructions also cause the malfunction, or simply an expected wrong answer.
- Make several physical copies of the malfunctioning section. Vary in several ways, adjoin copies and run together for efficiency.
- Reprogram for alternate methods of achieving the same result. This is often the simplest way to overcome blindness to the cause of malfunction.
- In difficult cases, change values of only one variable at a time for controlled experiment.
- Make full use of manufacturer-supplied tools such as de-flowcharting, dynamic testing routines, utilities, etc.

Additional treatment¹⁰ is available.

7. *Quality Control.* The best way to avoid malfunctions is to build software with quality controls applied during manufacture. All original programming, changes and additions to programs are done preferably in a computer-controlled environment.^{11,12} Such environments should be in general use by computer manufacturers by 1967, and should be available to users. ■

⁹ Univac, General Manual, Sleuth II for 1107, UP-3670, 1963.

¹⁰ Univac, P.I.E. Bulletin UP-3910.5, May, 1964.

¹¹ Bemer, R. W., *Software Systems Customized by Computer*, Proceedings IFIP Congress 65, Vol. II.

¹² Crowley, W. R., *A Possible Future System for Automating Control of the Development, Distribution and Maintenance of Programming Systems*, Proceedings IFIP Congress 65, Vol. II.

LAWRENCE G. TESLER
Information Processing Corporation
Palo Alto, California

Mr. Roberts and Mr. Flores Reply

EDITOR:

Mr. Tesler makes violent charges without pin-pointing the specific object he is attacking. For example, in his first paragraph he claims the paper contains "several serious errors." He never states anywhere in his letter what these serious errors are. What does Tesler mean? Does he mean the program philosophy is incorrect? Does he mean the program will not work? Does he mean the program can not handle the problem it purports to deal with? In the first paragraph he objects to our flowchart and program because it "erroneously claimed novel discovery of a standard method which removes some of its inefficiency, and failed entirely to remove its remaining inefficiency." Tesler never bothers to support his contention that the method is "standard" nor does he describe specifically these two types of inefficiencies.

Tesler's letter is inconsistent. For example in his paragraph 1 he implies that we erred in rejecting Berge's algorithm, while in his paragraph 5, he agrees that the Berge algorithm is not really useful for these problems.

Mr. Tesler's main claim appears to be that he can draw a "simpler" flowchart than ours. He fails however to emphasize the fact that he has borrowed heavily our logic and our ideas. For example, he employs the combinatorial matrix, the chain idea, and the ITEST array idea. See also our DO 100 loop. Prodded by our previous correspondence he modestly allows in his Paragraph 4 that his flowchart is based on ours. Admittedly our code, as any code, is susceptible to improvement. Mr. Tesler and also one of our colleagues have suggested testing each node for duplication in the ITEST array before testing whether the node is an initial point of a chain. This is a good idea which will simplify the handling of chains.

In our earlier correspondence and even now, Tesler fails to distinguish between the conception of an idea and the implementation of the idea by a programming language. For example, he rejects our claim of novelty for the ITEST array idea because ALGOL language has Boolean capability. The fact remains nevertheless that the concept of the ITEST array to reduce $(N - 1)$ comparisons to one comparison is novel. That Tesler can find an ALGOL 60 Boolean array to implement the ITEST idea (now that we have stated it) does not tarnish the novelty of the idea. After all, the idea came first, the implementation second. By Mr. Tesler's logic we should all be Shakespeare's since we have at our command all the words available to the Bard.

Mr. Tesler has noted correctly some minor errors in the flowchart. The corrections are initialization of $A(1) = 1$, when $ISTART = 0$ and recalculation of ITEST array when $ISTART = 1$. Column 11, of Table IIIA, should contain a 1 not a 53.

The intent of the "REMOVE CHAIN" box is to remove the chain and, of course, to remove the corresponding entries from ITEST, since this is in fact what the program carries out. Perhaps indexing in this block would have clarified this.

To conclude QGRAPH does in fact generate, as claimed, all the Hamiltonian circuits and their costs, consistent with the M -matrix and chains.

S. M. ROBERTS
International Business Machines Corp.
Houston, Texas
AND
BENITO FLORES
University of Houston
Houston, Texas

ESC Facility in USASCI

EDITOR:

The Morenoff and McLean paper (Jan. 67) is interesting as it demonstrates the Escape (ESC) Facility in the USASCI, which facility has not been exercised previously as it perhaps should. Certainly some code for non-numeric information processing could be permanently associated with one of the 255 indicators following the Escape character. It was proposed in 1962 in Stockholm that a block of these indicator codes should be reserved for association with alternate codes for programming languages. Subunits of such alternate codes might be identical to USASCI, but this is not vital.

In other words, the standardization process might, in the future, yield some alternate standard codes for special purposes which are in a sense subordinate to and linked through USASCI. Thus Mr. Morenoff's proposal is in no way in conflict with USASCI, and is indeed consonant with it.

However, the construction principles of his code are subject to scrutiny. One major premise is that functionally like sets of graphics (i.e., the alphabet) can be identified by bracketing between two binary numbers. For major sets this is also true in USASCI. A second premise is that within each such set "the ordering and sequencing of characters and words can be accomplished by simple binary comparisons of codes."

Here the author has fallen into a few traps, which might have been avoided by studying the bibliography of X3.2. The first is familiar to me because I did the same in the IBM 7030—that is, interspersing the upper and lower case representations of letters and thus giving each full graphic significance. Tain't so, and so the phone books show. With Mr. Morenoff's code, we would have:

De Carlo
De La Rue
De Long
DeLair
DeLancey
DeLaRue
Delancey
de Carlo
de la Rue
deLancey, and anguished subscribers.

The only proper method is to strip the case bit (b_6 in USASCI, b_1 here) and make a minor comparison upon equality. This may get a little more complicated in the case of italics, which obviously cannot be interspersed with the other graphemes in the proposed code.

Since the blank is high to digits and low to letters, we get:

A266 08
A2B and b8
A 66
AB66

The proposed code is perhaps more awkward than USASCI in comparing two numerals, since it requires radix point alignment and zerofill. Negative signs must be handled separately in either code.

There could be difficulties here with editing instructions.

Workers in the code microcosm will be sure to welcome Mr. Morenoff's interest.

R. W. BEMER
General Electric Co.
15430 N. Black Canyon
Phoenix, Arizona 85029

Am
techn
the r
foreign
condit
such
being
the s
often
Comp
this
exam
If
compu
less
some
inde
no f
and
ignor
consi
plicat
Are t
Even
draw
are
tees
eme
Fren
pend
also
ing o
orient
With
disre
it ma
are v
to le
out
of co
Europ
thing
are
(Se
have
of t
\$1,000
Volu

SOFTWARE AND REAL TIME SYSTEMS

Robert Bemer

"Software" ^{INCLUDES} the methods and the programs for the utilization of a computer system. It is a poor word, but it is intended to mean the opposite of the physical pieces of steel and electronics which are a computer's "hardware."

The interaction of computer components is becoming increasingly complex. In about 1957, with the advent of the *Fortran* language and processors, it became impossible to say that "the machine doesn't run," or, conversely, that "the program doesn't run." Today, in a real time system, one that is controlled by several remote terminals, it has become virtually impossible to tell what is at fault when something breaks down. Increasing complexity in computer systems, especially in the real time area, heightens the interaction between software and hardware. Thus, the previously separate functions of field engineering maintenance and software support tend to depend more on each other and even merge, particularly in the diagnosis of malfunction. What is necessary to avoid this complexity is to build a higher degree of reliability into programming systems and provide a more reliable service in the field when something does go wrong. It is advisable to use a computer network for fabrication, distribution and maintenance of software, particularly when there are multi-country sources for hardware and software system components, and a subsidiary structure for sales and service.

Communication Problems

Like other international computer companies, General Electric has a multiple source^s of hardware and software. The difficulties arising from this situation may be seen in the following hypothetical, but realistic, problem of Bull General Electric, which has several subsidiaries throughout Europe.

company, and its management. Computer, teleprocessing and mass memories are tools which have made an integrated management system possible, but they are not and can never be the system itself.

A long, detailed study of the information system itself is not necessary. In fact, when a decision system is established and the staff is well trained to make decisions, the procedures within the system automatically lead the computer to utilize all information necessary to suggest a decision. On the other hand, the possibility of asking the computer questions about any subject, and the existence of a data bank allow for a wide, flexible and economical use of available information, and greatly reduce paper handling, without the need for pre-established schemes.

To organize for an integrated management system means to plan all the steps necessary to bring the whole company to the right maturity to be able to use such a system. This is the longest and most difficult process and also the one which requires a steady guidance and definite plans and goals.

The establishment of a new, total management system is the major goal in developing an integrated system. The support of top management is a primary need in this process. Top management itself must work at this plan, because to transform a firm means, first of all, to transform its top management.

If an engineer in Dusseldorf working on a structural analysis problem has difficulties with his system, he cannot be sure if it is hardware or software that is failing. In Dusseldorf, he very likely has a General Electric computer for which the central processor was fabricated in Phœnix or perhaps Oklahoma City; the card equipment and punches, and perhaps some of the tapes, could have been fabricated in France; and the printer most likely would have been fabricated in Milan by Olivetti General Electric. In addition, the system he is using is probably composed of the work of programmers in Phœnix, Milan and Paris. The Dusseldorf engineer would report his problem to Cologne where, in turn, they report to Paris; there the problem is telexed on to Phœnix. In other words, with this sort of system, there is no way of diagnosing system breakdowns in a rapid-service form.

Communication Solutions

To remedy this situation, General Electric has devised both a short-range and a long-range plan to provide a fabrication and maintenance service to its customers by use of one or more central computers. In the production system now being considered, one or more central computers are connected by communications links to systems programmers, sales offices and customer installations. Motivations for such a system are

- optimization of European programming talent;
- rapid and efficient reporting and correction of malfunctions;
- common access files for sales control, queries and contracts;
- direct distribution of programs and documentation;
- controlled automated production of software;
- upgrading support personnel for better customer service.

In Europe, there are plenty of good software people, but they live in different countries. This is the crux of the problem. To fabricate a moderate-sized software system under present methods, between 50 and 100 people are needed in one place. In Europe this is fairly difficult because of language problems, social security, working papers, passports, homes, moving and the like. Europeans are less flexible for physical movement than are Amer-

icans. It is therefore expensive and difficult to set up a central programming site and staff it multinationally.

To solve these complexities, the General Electric plan is to set up a central computer in Europe connected directly to another central computer in the United States, each of which having direct communication to several programmers who are fabricating the software, to the salesmen and customer representatives who service customers, and to the field maintenance engineers. In short, the totality of people who have to do with the computers will do so through the central computer.

It is possible for a number of people working thousands of miles away from each other to produce software jointly, since software is a largely intellectual product. By its very nature, it requires centralized control for fabrication, largely because of its relative invisibility. But it is not especially critical where the actual programmers are located with respect to the other functions of the computer manufacturer, subject to sufficient liaisons with designers. One characteristic of time-sharing systems which was not expected by too many people is the now-proven, intense human interaction possible between people connected to a computer—and now this interaction will actually fabricate the software that drives the computer itself.

International Software Development

Preparation for this type of work is presently under way in London, where we have assembled a multinational group to build a major software system. In many ways the methods remain the same as before. However, the group is headed by an American team of absolutely top rank which is enjoined to:

- teach all techniques of good software production while the project is in operation;
- demonstrate all the techniques of good software project management used, with particular attention to costs and meeting of schedules;
- make available to every participant the schedules and proposed costs, so that they may simulate project management as in a case study;
- indoctrinate in the usage of the new General Electric docu-

mentation models, that maximum information will be made available to prospective users at the earliest moment, in a necessarily complete and standardized format.

These Europeans are learning here, so that when they go back to their various countries, they will form a nucleus of these ~~eventual~~ people who will fabricate software remotely. Essentially, then, this is a seeding project to develop talent in the various European countries. This group is doing a complete software system for a 400 computer which operates on a disc only—it doesn't require magnetic tape and the software system itself primarily uses the disc as its home base or residence.

It is of interest to know that in the planning stage of this project, which was carried out by three of the American team and four from France, over 700 pages of specifications were produced in six weeks; the Phœnix organization called this the most remarkable effort of this type that it had ever seen. Thus, we hope to make a graceful transition in our subsidiaries, upgrading our personnel to full software construction capability, while still proceeding cautiously enough for adjustment when goals are missed. The time scale is not felt to be critical, as there are no expectations that European communications networks will all be of sufficiently high and uniform quality for this type of operation for at least three more years.

The second thing General Electric is doing in the way of preparation are certain experiments in remote communications here in Europe. For example, there is a GE 600 in Sweden, which is soon to be operated remote from the 115, which is fabricated in Milan. Another experience successfully tested at the London Data Fair in early 1966 was to operate a New York computer from London.

The third element of preparation is something recently developed in Phœnix, called the 6.45 System. The common method of correcting programming mistakes is to punch cards and collate these against the magnetic tape, producing a new tape which is the corrected program. But now programmers for this system use teletypes, often in their own homes, having developed a general string manipulation system. This system takes strings of characters, and can add to it to form the original string, copy it, delete elements of the string, find elements in the string and replace them. This is done under a program called ACE—Automatic Context

Editor. There is also a protection program called ACCESS. The machine asks the user number, then the file name, then the password. Persons without the proper password are denied access to the file, thus guarding its information.

An example of an application of this technique is for organization charts. When somebody's salary or status is changed, the manager marks the file from a telex, and gives access to this file, with its changed condition, to the payroll man. It can also be used for intraoffice communications. Each file also has "mailbox" as a sub-file. Whenever access is made to a file, the first thing that comes out is the latest mailbox information. In this way, people who use a particular file can communicate directly about and through it.

Future Implications of the System

It is certainly possible to make a superstructure of this system, where the meaning, the relevance of what one wishes to put in, is filtered and controlled before it is accepted. For example, before a programmer can actually enter his instruction into the machine for testing, it must pass a certain acceptance filter within the computer. At the end of the month, it is easy for a supervisor to check the records to see how many instructions each programmer got by the filter, thus discovering his productivity. In short, the computer program itself can thus be made to be a more effective supervisor of programmers than humans.

Such systems are certainly interactive to the extent required for intercommunication in joint remote production of software systems. Furthermore, software tools may be constructed for the central computer which monitor and control this production process, filtering out unacceptable, incompatible and non-standard inputs in a more effective way than possible for the human programming supervisor. Human supervisors cannot tell what programmers are doing by looking at the programs; work is shown only in running them. Therefore, control systems for software production are being installed so that programmers must meet certain benchmarks. Then a situation can be envisioned where there are several programmers linked to each other in the production of the original software. After it is made, it must be distributed, improved upon, and maintained free from malfunction;

the documentation must be complete and accurate, and the customer must be taught how best to utilize his system.

A remote terminal is assumed to be a necessary adjunct to every computer installation, existing either on site or at the sales office. It is also assumed that qualified systems programmers are engaged in remote production of new software, while servicing current software and assisting customers in its proper usage.

When a field engineer on site is confronted with a malfunction which he cannot immediately correct, he reports at his terminal that a particular software element has malfunctioned while using a particular user program. This report goes to the central computer, which looks up the responsible programmer, locates him, and asks him to correct it. He then studies the program, and has the ability to access other computers of that class. The net effect of this is to have the original programmer theoretically on the site where the problem is.

When the programmer makes a change, he also has to go back to the system. Before the change can be distributed, it must go through a documentation check, so that at any time the program and the documentation explaining how to use it are accurate and matching. After this check, changes on both program and documentation are distributed to all users of that class of equipment. This means that the computing installation of the future will always have a remote terminal which accesses this central support computer.

A system for automated software production requires complete files on equipment installed and on order; information must be complete on both hardware and software configurations, software production and delivery schedules, authorized software units, and associated manuals. A sales office connected by terminal unit to this central system may therefore verify accuracy and availability of the items included in its sales proposal. Normally such basic information will also be distributed to the field sales force in the form of a price list or sales data book. This easy access to the single, current, official information avoids errors and loss.

This system has some other advantages besides the production of software. Incorporated in the system are schedules of production, which, for example, may include hardware production, deliveries to the customer, and legal and non-legal configurations. Thus, the sales office gets the most accurate information possible on what it is selling. Direct connection to sales offices should

facilitate usage of computer programs to evaluate computer performance. Through the SCERT program, it is also possible to evaluate not only the computer a salesman is offering, but also competitive computers on the same customer's problem.

A particularly important application of this system is the ability to set realistic delivery schedules. The central system provides the actual expected schedules, together with a margin for contractual acceptability. If schedules in a proposed contract fall beyond this limit, then the salesman is authorized to conclude the order without further approval in this area. Thus the system can serve as an order acceptance filter and update the on-order file. It should not be too difficult to transmit this file or changes to the factory to use as the official basis for factory order and production schedules.

One of the best features of this system is the mobility of programmers using the remote terminals. By having talent decentralized in various strategic areas around Europe, programmers are physically available to assist customers. There are three classes of work that can be done: original creation of the basic software, maintenance of the software, and assistance with customer problems. One of the advantageous features of the remote production method is that field reporting and maintenance may be done by experts who reside physically near to customer installations. Since this is only a part-time assignment, the balance of their time may be spent in creative software production and even research. Furthermore, working directly with customers yields a closer association with the mix of actual problems to be solved. This experience is very valuable in the design and production of new systems, for the latest needs and interests of customers can be represented in both the design and quality assurance testing phases.

The last benefit to be mentioned is that of optimization. The central system maintains all schedules; the contractual requirements are grouped and associated with the various hardware and software units. This information may be used to manipulate production priorities to give optimum customer satisfaction.

Summary

To summarize, the computer, by serving as a buffer and a filter between programmers building software remotely, can serve better

than a human supervisor. Programmers will be placed under strict production control, and the customer will get rapid correction by having competent personnel near his site. It will be as though, when an automobile breaks down, the chief engineer of Simca were there to repair it. This is an ultimate goal of customer service.

Toward Standards for Handwritten Zero and Oh

Much Ado About Nothing (and a Letter), or A Partial Dossier on Distinguishing Between Handwritten Zero and Oh

R. W. BEMER*
Member, ACM Standards Committee

The Chairman of the ACM Standards Committee, Julien Green, has charged me with making "more effective use of *CACM* for communication . . . to get grass-roots opinions from the ACM membership." This paper is the first attempt.

A partial dossier on distinguishing between handwritten zero and the letter oh is assembled here. This presentation was triggered by a request for guidance in this matter presented by the United Kingdom Delegation to ISO/TC97/SC2, Character Sets and Coding, at the meeting in Paris on 1967 March 13-16. The matter is just now in the province of USASI X3.6, to which comments might be directed.

Comments will be expected within sixty days [by approximately October 1st].

58SEP11

H. S. BRIGHT, TO THE MEMBERS OF SHARE [also in *CACM* 2 (May 1959), 9]

"Letter 'o' is larger than '0' (zero) and includes a flourish at the top as when written as an English script capital letter. These distinctions are, clearly, a matter of personal taste. Better ones may well be proposed; in any event, I feel that some obvious distinction should be made, and that the presently common use of 'Ø' to represent the letter 'O' is confusing to the uninitiated and is harder to write than 'o'."

59MAR11

R. W. BEMER, PROPOSAL FOR A GENERALIZED 256 CHARACTER CARD CODE SET [also in *CACM*, 2 (Sept. 1959), 19-23]

"OH" is shown with an interior dot.

* Mr. R. W. Bemer is the member of the ACM Standards Committee charged with responsibility for interfacing with the ACM membership on current standardization activities. His address is: General Electric Co., 13430 N. Black Canyon, Phoenix, Arizona 85029.

59AUG26

G. R. TAIT, RPQ TO IBM [in SHARE Secretary Distribution 58]

"This will confirm my recent phone call for a 720A RPQ. Specifically, the RPQ desired is one that will permit distinguishing the alphabetic character "O" from the numeric digit "zero". I would like to know the cost of adding two more control rods to the alphabetic "O", namely, rod 5 in row one and rod 4 in row two. This RPQ would yield an alphabetic "O" that would have four rods in the upper right-hand corner printing as a group. As such it would be similar to an inverted "Q".

60JUN1

AN EXTENDED CHARACTER SET STANDARD [IBM Report TR 00.721, by R. W. Bemer and W. Buchholz]

"A long-standing source of confusion has been the distinction between upper-case Oh (ø) and zero (0). Some groups have solved this by writing zero as Ø. Unfortunately, other groups have chosen to write øh as ø. Neither solution is typographically attractive. Instead, it is proposed to modify the upper-case øh by a center dot and to write and print it as ø whenever a distinction is desired."

60NOV4

R. W. BEMER, IBM, TO BRIAN POLLARD, CHAIRMAN ASA X3.1

"Although not the most immediate problem of your subcommittee, standardization of an effective means of differentiating between the alphabetic "oh" and the numeric zero is receiving much attention. Strictly speaking, only a part of this problem is in your province. Nevertheless, it would be beneficial if you could effect an early solution.

"I shall outline here some of the background information as I know it. Undoubtedly more complete information is available through certain competent individuals, whom-ever they may be. A wide distribution of this letter should serve as an impetus to elicit this information.

1. *Differentiation by Context.* Most standard typewriters make no distinction between the "oh" and the zero. In fact, some do not have a key for the zero, relying upon the capital "oh" instead. This is a convention rooted in time. We speak, for example, of the Boeing seven-oh-seven, not the Boeing seven-zero-seven. We distinguish the two usages by the context. If inbedded in a set of alphabetic characters, it is assumed to be an "oh". If inbedded in a set of digits, it is assumed to be a zero. It is precisely this need for contextual differentiation that causes trouble in data processing equipment. Many fields are mixed alphabetic and numeric, such as part numbers, storage print-outs, etc.

Differentiation by context is not suitable to any form of character recognition.

2. *Differentiation by Shape.* A great deal of handset type and some typewriters (for example, the IBM Executive) differentiate by making the "oh" fatter and rounder while the zero remains a narrow oval. Some fonts have distinguished even more by making the "oh" squarish. Such a difference in character width affords partial distinction; it requires a size discrimination from the reader which requires more concentration than graphic discrimination. As a solution, it is inadequate in two respects: (a) for typewriters with constant spacing, and (b) for parallel (or line at a time) printers. Differentiation by shape is hardly suitable for character recognition.

3. *Differentiation by Additional Marks.* It is common practice (for both letters and figures) to aid identification by adding dots, small circles, underlines, overlines, slashes, umlauts, etc. If properly used, this seems the best method for character recognition.

3a. *The Slash.* The slash has been used to distinguish between the "oh" and the zero. The only problem is that users have been inconsistent in deciding which of the two should be slashed. To mention an example from either side, the Signal Corps has used a slash through the zero for many years and this is found on many teletypewriters; the SHARE organization (a group of users of IBM 704, 709/7090 equipment) uses the convention of a slash through the "oh" for at least the handwritten symbol (while not a SHARE standard, many installations have their high speed printers thus equipped).

A case may be made for either convention on the basis of usage. Technically, however, it would seem that the slashed "oh" is preferable. The Waverly Press Type Manual shows slashed zeros in their font on page 66, but *only* when the decimal digits are similarly slashed. To the contrary, pages 50 through 56 (accented letters) show the "oh" slashed in most fonts as well as C, D, S, etc. Historically and mathematically the practice has been to slash the alphabetic character because of its lesser frequency compared with numerics, (the mathematician's handwritten Z, for example, which is distinguishable from the digit 2). A possible conflict exists with the Danish and Norwegian languages, which utilize both the slashed and nonslashed "oh". However, reasoning from the German typewriters which have special keys for each of the letters A, O, and U, in the umlaut form, one may assume this is merely a convenience in contradistinction to having a special umlaut key that would require backspace and overprint. The slash here seems to serve a function similar to the umlaut, and thus the slashed "oh" can really be regarded as similar to an unslashed character.

3b. *The Tail.* Some output printers use a degenerate slash in the upper right-hand corner of the "oh", as in an inverted Q. A specific example is that at the Western Data Processing Center at UCLA. Furthermore, the handwritten capital "oh" has a similar tail (ø).

3c. *Interior Dots.* According to Mr. Julius Agin of RCA, their numeric scanning typeface was derived from a report of the US Army Ordinance Corps, TR-39, 15 January 1954, entitled "Standardization of the 5 X 7 Font". The zero originally had a short vertical stroke in the center which was "modified to the two dot center to help differentiate it from the numerical 8 and thus further increase the distance between the ten digits for improved reliability in a numeric-only machine". I understand that both the Stromberg-Carlson printer and typewriters of the Armour Research Foundation utilize a single dot interior to the zero. This would appear to be contradictory to character recognition requirements, since this would make the narrow zero with the dot even less distinguishable from the 8. The printer for the IBM 7030 uses a dot interior to a fat "oh". This was chosen because it seemed impossible to reconcile the argument between slashing the "oh" or the zero and because it seemed reasonable to make the less frequently printed character have the special marking. In addition, this convention does not conflict with any existing usage of accented letters. This convention has also been used miscellaneously in ornate lettering, although in no way connected. An everyday sample is that of the advertisement for Original Pabst Blue Ribbon Beer.

"I would appreciate your giving this letter (or some augmented version of it) wide distribution for feedback purposes."

61MAR23

MINUTES OF SHARE XVI

Page D.20.3 shows a proposed SHARE Character Set (King) where the "Oh" has an interior dot.

61MAR29

G. F. RYCKMAN TO SHARE COMMITTEE CHAIRMAN, A PROPOSED STANDARD FOR SHARE CHARACTER SET [In SHARE Secretary Distribution 82]

"... the General Standards Committee first agreed that the present 48-character set must remain intact. One exception to this is the alphabetic O, which is often confused with the digit zero. The recommendation is that the letter O (as printed) be changed to distinguish it from zero, e.g. ø."

61AUG21-25

MINUTES OF SHARE XVII

"George Ryckman (GM), General Standards chairman, called the attention of the members to the new proposed 61-character set which appears in Appendix E.3.1. . . . In answer to a question by Frank Wagner (NA), Tom Steel (RL) explained that the zero would differ from the O in being noticeably thinner; no extra stroke would be added to either character."

62MAR28

H. MCG. ROSS, FERRANTI, LTD., TO R. W. BEMER, IBM
"We are now experiencing a significant amount of difficulty resulting from confusion between the letter O and number zero, and we have got to do something about it. My first reaction was to write to you to enquire whether you have

settled in your own mind a preferred solution to this problem, or whether there is some arrangement which is becoming fairly widely accepted.

"I might explain the way our own thinking has been going. We feel that, as far as any printing machine is concerned, it would be adequate to use for letter O a rectangular shape with rounded corners, and to use the ordinary elliptical shape for zero. In reading printed documents, we have found it rather irksome to have a dot in the middle of a letter, and we feel that a full-length solidus through every zero looks ghastly.

"The distinction will not, however, be sufficient for handwriting. For this we would suggest the convention that the letter and number should each have its own distinctive mark, but that these should only be written when there is any possibility of confusion. The distinctive marks for this handwritten form which we at present favor are a dot in the letter and a little mark coming down from the top right corner of the figure zero.

"Thus, the conventions would be:

	Letter	Zero
Printed:-	□	0
Handwritten	(only when a possibility of confusion):-	
	⊙	0̣

"In connection with zero, we understand that the full solidus mark has recently been adopted by some British Government groups and also by ICT, but this does conflict with its use as phi and with the Norwegian letter. We feel our suggestion looks better and would convey to the user the same meaning as the full solidus mark. We feel that the dot and the little dash would be very easily written by hand."

62APR6

R. W. BEMER, IBM, TO H. MCG. ROSS, FERRANTI LTD.
 "I enclose a copy of a previous letter (to Pollard, 60 Nov. 14) on this subject. As far as I know it has not done any good although I have been insisting that it is a proper subject for standardization bodies. You are, of course, at liberty to use this letter in any way that you wish.

"You can see that I concur with your view that the solidus is out of the question. However, I cannot agree fully with your solution. The usual mathematical convention is to place an additional mark or serif on any letter that might be confused with digits, although this is sometimes done within the digit group as in the German 7. For this reason, and the additional reason that the script Oh customarily has a tail, I would prefer that the letter O carry the distinguishing mark in the handwritten form and that the 0 carry no mark. In this case either of your conventions would be suitable for the letter. I do prefer the center dot and have not found it irksome to read. I do not

think you will find it so in the attached copy of output from the printer of the 7030 system.

"I would agree to any conventions for printing where there is more area enclosed in the letter Oh than in the digit 0. Your squarish solution conforms to this principle and so does the typewriter convention of having a fatter oval for the letter."

64MAY26

G. CARLSON, BRIGHAM YOUNG UNIVERSITY, TO J. E. FEELY, THE MARTIN COMPANY [In SHARE Secretary Distribution 122]

"I would appreciate the assistance of the SHARE members to clarifying some conventions in program writing. I would appreciate it if you could publish the attached questionnaire:

In an attempt to clarify hand-lettering standards, I would appreciate your marking the following questionnaire.

CHARACTER	HAND CODING CHARACTERS AT YOUR INSTALLATION (Please circle the one used)	
Zero	○	⊙
Letter "O"	○	⊙
Number one	1	1̣
Letter "I"	I	Ị
Letter "Z"	Z	Ẓ
Number two	2	other

64JUN14

SHARE PL/I PROJECT TO BEN FADEN [In SHARE Secretary Distribution 153]

"At the PL/I Project meeting in New York on June 7th, the problem of character sets and how the various configurations should be handwritten was discussed. We decided on the following:

minus sign	-	underscore	—	
letter	I	number	1	OR ⊥
letter	Z	number	2	
letter	⊙	number	○	
AND	⊘			

The slashing of the letter Oh is in conflict with the IBM standard. I am enclosing a copy of a letter from Elliot Nohr (IBM). Before we can recommend further action to IBM on this, we would like input from the rest of SHARE."

64DEC

IBM CORPORATE SYSTEMS PRACTICE 2-8015 [In SHARE Secretary Distribution 153]

INTRODUCTION

"1.1 Objectives. The purpose of this practice is to pro-

mote a consistency in the representation of zero and a consistency in the representation of oh to avoid misinterpretation.

1.2 *Scope.* This practice establishes the methods of making distinguishable the graphic number, zero, and the graphic letter, oh.

1.3 *Applicability.* This practice is to be followed where it is necessary to differentiate between zero and oh and where misinterpretation may arise, as, for example, in hand drawings, documents for keypunching, and printing and visual display devices. Where any other practice is employed, transition to this practice is to be completed prior to December 1, 1965.

The character shapes for the stylized fonts for OCR and MICR, including zero and oh, are defined in separate standards documents. The OCR graphics shown in this practice are for reference only.

RECOMMENDED PRACTICES

2.1 It is the existing and preferred practice that, on devices such as printers and typewriters, the character shapes for zero and oh be sufficiently different and distinguishable. Except for the stylized fonts used with OCR and MICR it is the preferred practice to provide a narrow zero and a wide oh. Examples are:

	Zero	Oh
Typewriter (engraved-type font)	0	0
Keypunch (matrix-type font)	0	0
OCR	□	◇

2.2 If the method described in Paragraph 2.1 is not feasible for either handwriting or printing and a slash is used to make zero and oh distinguishable, the practice to be followed is:

The number, zero, will be slashed, i.e. 0.

The letter, oh, will not be slashed, i.e. O

2.3 If neither of the methods in 2.1 or 2.2 is feasible and some line or mark is needed (as opposed to character shape stylization) to make the zero and oh distinguishable, the line or mark is added to the zero."

65OCT25

HARVEY S. MILLMAN, TO THE SECRETARY OF ASA X3

"With all the work being done on character sets for magnetic encoding, optical scanning, etc., we wonder if there is a convention to aid the lowly keypunch operator.

"The problem a keypunch operator faces in trying to distinguish letter O and numeral zero, letter I and numeral 1, numeral 2 and letter Z is well known. Existing conventions are in conflict.

"Does an ASA Standard exist for "Handwritten Characters for Keypuncher Recognition"? If not, does ASA plan to propose one?"

MINUTES OF BUSINESS EQUIPMENT MANUFACTURERS ASSOCIATION, DPG STANDARDS MEETING

X3.2 Proposal for Standard Methods for Representation of Similar Graphics in ASCII

X3.2 reports that there is a problem in the graphic shape representation of 1 and I, 0 and O, and 2 and Z. They have asked the Chairman of X3 to determine whether or not this problem is within their scope. It was suggested that the Chairman of X3 form an ad hoc group of all of the subcommittee chairmen and determine what should be the method of distinguishing between similar graphics in ASCII.

Mr. Vidro commented that originally precautions were taken not to specify the graphic representation of the symbols in ASCII and leaving the stylizing of symbols to the manufacturers.

It was noted that since the problem is more connected with the handwriting of the symbols than with their printing, the work might more properly be done in X3.6.

65DEC14

MINUTES OF USASI X3, 21ST MEETING

7.5 Standard Representation of Similar Graphics in the ASCII

... A motion was unanimously approved that:

X3 refer the problem to X3.6, making them principally responsible, with liaison and participation by X3.4.

66JAN

L. RICHARD TURNER, NASA LEWIS RESEARCH CENTER, LETTER TO EDITOR OF CACM, "ON THE CONFUSION BETWEEN "0" AND "O" "

"I should like to describe briefly a technique which has been in use at the Lewis Research Center of NASA for approximately ten years for resolving the confusion between the mark 0 intended to mean zero and the mark O intended to mean the character between N and P in the Latin alphabet.

"As applied to the management of identifiers and numerical values in assembler or compiler languages, it has worked without failure and does not require that human programmers differentiate between the similarly shaped symbols for zero and the letter "O".

"The technique consists of translating the code for the letter "O" to the code for the numeral 0 whenever it is encountered in the input character string. If the string consists only of items such as numbers and names and it is necessary to sort alphabetically on names, the occurrence of an alphabetic character within a name field is used to cause the code for zero to be retranslated to the code for the letter "O" by a rescan of the characters in the name field.

"If no sorting is required, the retranslation can be avoided, provided that delimiters such as FORMAT or GO TO are spelled with zero within the recognizer segment of a translator. It is also necessary to redefine

identifier as

(identifier) ::= <letter> | <identifier><letter> | <identifier><digit> | <0><identifier>

where it is understood that the letter "O" is removed from the standard definition of letter as in ALGOL 60. The redefinition permits the inclusion of identifiers such as ODD or OOPS but prevents the use of an identifier consisting only of the repeated mark O.

"This technique requires consistency of use and might result in chaos in a warehousing operation in which the letter "O" is used in parts labels with check digits."

66MAR4

PURPOSE, SCOPE AND WORKING PLAN OF X3.6.5

A fourth group on zero, 0, etc. has not yet been formed.

66MAR14

MINUTES OF 26TH MEETING OF USASI X3.6

X3.6.5.4—TG to study common practices for distinguishing between zero and O, 2 and Z, etc. W. Morgan agreed to act as temporary chairman.

66APR26

MINUTES OF 27th MEETING OF USASI X3.6

2. The Purpose, Scope and Working Plan of X3.6.5.4 is suggested as:

- a. *Purpose:* To promote consistency in the presentation of 36 alphameric characters in the definition and man-to-man communication of information processing problems.
- b. *Scope:* To establish a method for unmistakably presenting the alphameric characters A to Z and Zero to 9 in man-to-man communication.
- c. *Working Plan:* (1) Collect data on common practices for distinguishing between zero and letter O, one and letter I, letters u and v, etc. (2) Evaluate the data collected. (3) Prepare resolution for X3.6.5 on feasibility of establishing an American Standard in this area.

3. A suggested title for X3.6.5.4 is "Alphameric Presentation".

66MAY26

MINUTES OF 28TH MEETING OF USASI X3.6

Alphameric Presentation. Members of X3.6 are urged to send existing standards and other information to Mr. W. D. Morgan. Task group members are needed.

66MAY

MICHAEL L. PERSHING, UNIVERSITY OF ILLINOIS, LETTER TO EDITOR OF CACM "ON 0 AND O"

"In reading the letter by Mr. Turner [On the Confusion Between "0" and "O", *Comm. ACM* 9, 1 (Jan. 1966), 35], I notice that his redefinition of the ALGOL <identifier> fails to allow those such as "012ABC", which contain a digit immediately after the zero. It seems that such a combination of characters will pose no major recognition problems, and should be allowed, providing only that it contains a letter somewhere, other than the initial "O", which may logically be taken as a letter or a digit.

"I therefore offer an addition to Mr. Turner's redefinition:

(identifier) ::= <letter> | <identifier> <letter> | <identifier> <digit> | 0 <identifier> | 0 <unsigned integer> <letter> |

"I personally have avoided most of the confusion between the two characters by attempting never to use either character in mnemonic symbols unless it has some mnemonic significance and it is difficult to take it to be the other. There are, however, some situations which are beyond the control of the programmer, but with which we are made to live; two of these are the FORTRAN internal functions, MAXOF and MINOF. For situations such as these, Mr. Turner's solution seems somewhat appropriate."

66JUN20

H. W. NELSON, CHAIRMAN OF SHARE CHARACTER SET COMMITTEE, TO BEN FADEN, NORTH AMERICAN AVIATION [In SHARE Secretary Distribution 154]

"I strongly support the recommendation of the PL/I Project that one should slash the letter "O" and leave the number zero unchanged when they are handwritten. Numbers are handwritten much more frequently than letters on sheets to be keypunched and therefore letters, rather than numbers, should have extra lines added to make them distinguishable. In our company we have been applying this rule of adding extra lines to letters with complete success for the past 15 years in preparing sheets for keypunching. After all, this is simply an extension of the rule for slashing Z's that we all learned in mathematics. Also, if one is not especially careful, a slashed zero may look like a six or a nine to keypunch and verifier operators.

"IBM's procedure of slashing zeros is a "recommended practice," and not a standard. Furthermore, as far as I am concerned, it is applicable only internally to IBM. It, as such, should not become by default a *de facto* standard outside of that organization!

"I find it especially disconcerting to see slashed zeros in all the written examples in "IBM System/360 Operating System FORTRAN IV(E) Programmer's Guide," C28-6603-0. (It would have looked even more ridiculous if they had properly filled in columns 73-80 with sequencing information instead of leaving them blank.) This, I fear, will lead to a lot of unnecessary confusion in setting up job control cards. I also question ending the name of the FORTRAN compiler with a zero. Since the rest of the name is letters, I'm sure many people who are not especially sharp eyed will use the letter "O" instead. Frankly, using OS/360 will be tough enough without adding all these stumbling blocks besides. When is someone at IBM going to start obeying those "THINK" signs they have hanging up all over?"

66JUL22

GEO. WIEDERHOLD, STANFORD UNIVERSITY, TO H. W. NELSON, CHAIRMAN OF SHARE CHARACTER SET COMMITTEE [In SHARE Secretary Distribution 155]

"An escapist solution that we have used for a long time is overlining the letter O thus: \bar{O} .

"I agree with your reasoning, but I hate to see opposite conventions used by groups that have to work together."

66SEP27

MINUTES OF 29TH MEETING OF USASI X3.6

X3.6.5.4—Alphameric Presentation—a letter was sent to all Army Departments asking for information on their practices. Some information has been collected from other sources and members of X3.6. A letter survey to be sent out by BEMA will be prepared. This task group will meet with the next meeting of X3.6.5 working group. Members of X3.6 are urged to join this task group.

66OCT14

R. W. BEMER TO J. W. PONTIUS, X3.8

"I am attaching some previous correspondence for your information. In addition, Mr. Kivett has indicated that he does not favor using the slash in publications (G.E.) for external distribution. Instead, the ratio of width to height should be the distinguishing characteristic, as specified in item 2 of my 1960 November letter. While Director of Systems Programming at Univac I invoked this same policy in publications to avoid confusion with conflicting practices.

"I think we may conclude from this material (in particular the OCR work) that it is not likely that there will be a national or international standard concerning the slash and other distinguishing marks. A convention may be maintained for hand-written material in the case of keypunching. This may vary locally, but since the information should not be exchanged in the hand-written form this does not seem serious. We are obviously progressing to a proliferation of keyboard devices where the

originator produces his own hard copy; thus the middle man is eliminated and conventions become less necessary."

(In reply to a query on the following, of unknown source): "Europeans, U. S. Military, and most U. S. industrial concerns use the slant mark on the 0 (zero) when needed to distinguish it from the O (oh). One way to avoid ambiguity is to adopt the rule that a horizontal mark signifies a letter, hence Z, and Θ ; and the slant mark to indicate a numeral, hence \emptyset . (This may conflict with the European 7 (seven) but not many of us cross the seven.) This rule is primarily for use on hand-written and draft typed material. Wherever possible our published material should be prepared with typewriters with distinct circles to represent the letter O (oh) and distinct ovals to represent 0 (zero). Every effort should be made to keep from using any marks to distinguish the \emptyset from the θ in illustrations or text material of our external publications."

66OCT26

MINUTES OF 30TH MEETING OF USASI X3.6

A letter requesting information from the EDP industry is to be written by W. D. Morgan within the next week. The letter will be sent out by BEMA to X3 members, and other EDP users in the industry. A response time of 90 days will be requested.

66DEC7

MINUTES OF 31ST MEETING OF USASI X3.6

TG4—Alphameric Presentation—a letter requesting practices in this area was sent to BEMA and distributed by them to X3 members. Some concern was expressed as to whether this represented the entire information processing community.

SOFTWARE INSTRUMENTATION SYSTEMS FOR OPTIMUM PERFORMANCE

R. W. BEMER and A. L. ELLISON

Information Systems Group, General Electric Company, Phoenix, Arizona, USA

49

Since the inception of electronic computers we have been accustomed to instrumentation of the actual hardware by the common devices of the electronics field - counters, oscilloscopes, voltmeters and such, and now perhaps laser devices. However, there is an amazing phenomenon in the almost complete lack of instrumentation for software. Yet we are aware that there are two levels of software imposed upon the hardware. First the basic system of the supplier and then the user-constructed application. Performance degradation is the result of this uncontrolled and unmeasured imposition, with efficiencies running to as low as 5 or 10 per cent of the theoretical potential of the system, as the hardware is capable of driving it.

How can this state of affairs have existed so long without insurrection by the users? Simply because all systems available have such problems to a greater or lesser degree, and the inefficiencies, gross though they may be, occur in very small time intervals for each function. The programmers for both suppliers and users cannot detect that something required ten milliseconds to do that should have taken only one millisecond, for humans cannot relate to anything that small in real time. Similarly they cannot detect the operational malfunction, which ends by doing the apparently right thing although it may have gone through a thousand improper paths. Instrumentation must be used to detect what is *really* going on in the system, with system-oriented advice and tools to enable one to correct faults or choose new options.

Such instrumentation should be used by the supplier to optimize the system software for the most likely usage by a spectrum of users. Then it should be used by the application installation to measure whether inefficiencies are introduced by the specific way it uses the system (usually quite far from what the supplier had envisioned). Options to modify the system must be provided, such as choices of algorithms or priorities. Finally, instrumentation should be used periodically to monitor the effect of current problem mix (particularly for multiprogramming and multiprocessing environments), to indicate how to vary these parameters on-line to accommodate the mix better. Yes, software process control.

It is the purpose of this paper to expose the problem, justify instrumentation in the software form (not just hardware), give some theory of complex processing environments, and give examples of actual inefficiencies which can be detected only with suitable instrumentation. Implications for future design of hardware and software are discussed, together with lessons we believe we have learned in the development.

1. INTRODUCTION

Instrumentation should be applied to software with the same frequency and unconscious habit with which oscilloscopes are used by the hardware engineer.

Software attempts to keep the hardware running as close as possible to maximum potential by minimizing idle time and bridging the interferences between successive (and perhaps simultaneous) programs of different character and facility requirements. Software overhead must not degrade the system so much that only a slight amount of the total time and resources are expended in the production of answers. When a system operates at 25% efficiency, as many do of this third generation, there is a 4 to 1 leverage on system productivity. The choice is clear between instrumentation and improvement or buying three more systems. Efficient software is a tacit requirement in computer systems design.

In its absence, the balance of the system may be distorted seriously.

Program testing consists of measuring a program's performance against planned goals. Does it really work as designed? We are accustomed to expect a new program to fail in various ways until several iterations of a test and correction process have been carried out. When the program is "debugged", a certain measurement of quality has been made. However, the actual efficiency is rarely measured against the design. Thus a very important measurement of quality is *not* made.

Efficiency can be defined only in terms of measurement and statistics. The programmer's intuition (which we hope fervently will lose respectability and justification) does not suffice in a complex environment. Gross measurements, analogous to the "idiot lights" of automobiles, are not sufficient to show why proper efficiency is not attained. A gauge is required, as well as

special instrumentation systems at the service center. The various modules must be timed individually, and their interactions measured individually.

If the actual performance of the system is less than planned goals, the manufacturer should know whether the system is capable of being improved to the proper condition, and the user should know if the inefficiencies stem from his job mix, his methods of using the equipment, or inherent system inability.

Without instrumentation, the user is swimming against the tide of history. It is commonly thought that a good programmer naturally achieves at least 80% of the maximum potential efficiency for a program. But while systems have increased greatly in size and complexity, the average expertise of programmers has decreased. In fact, it is axiomatic that virtually any program can be speeded up 25 to 50% without significant redesign! [1] Unless monitored and measured, a program's efficiency may easily be as low as 25%. What is worse, multiprogramming, multiprocessing, real time, and other present-day methods have created such a jumble of interactions and interferences that without instrumentation it would be impossible to know where effort applied for change would yield the best return. One tries to mine the highgrade ore first, while it still exists.

2. THEORY

The potential causes of inefficiency are closely analogous to those in human procedural systems:

waste	inadequate resources
indecision	poor resource deployment
interferences	high overhead

Inefficiencies of all types can occur at various levels within the computer system.

Until the software has been instrumented and measured, there is no justification for any belief that severe inefficiencies are absent. Furthermore, it will be evident in the following section on *Practice* that the specific sources of inefficiency can rarely be identified without instrumentation. A detailed description of the analysis process is given in ref. [2].

The various kinds of instrumentation applied in efficiency analysis can be categorized as:

- gross distribution of device times within a program
- fine distribution of processor time within a program

- workload statistics from installations
- traces
- instantaneous distribution of system resources
- overall time distribution of system resources

Improvements based on instrumentation and analysis take two forms: elimination of efficiency errors and optimization of system parameters. Lack of optimization in an uninstrumented system must be regarded as one type of efficiency error.

Optimization can aim for the general user, a class of users, or for a specific user installation - or even for temporary conditions (perhaps cyclic) within an installation. Optimization by the supplier will almost undoubtedly aim for the hypothetical general user, perhaps with assistance provided for further on-site optimization. Unless the system is optimized for the individual installation, its full potential cannot be realized. Instrumentation is of course necessary to identify the parameters to adjust and in what degree.

As software instrumentation gains acceptance, it should be designed into the programs from the outset. Efficiency measurements are then available from the early stages of checkout, and tuning proceeds as an integral part of the diagnostic process. We hope to see more instrumentation for the design process itself in the future - high-level system modeling, with simulation used to measure key design choices.

3. PRACTICE

The proving ground for many of the instrumentation ideas described here has been the GE-625/635 computer system family, a large-scale machine line oriented to multiprogramming and multiprocessing.

It must be emphasized that instrumentation and analysis are entirely symbiotic. Accordingly, instrumentation techniques were developed in the course of a broad-spectrum efficiency analysis of the software. The analysis has progressed from the operating system through the language processors to the user's application programs, with efforts well rewarded at all stages.

We give here a catalog of specific contributors to inefficiency, derived from the authors' experience and in some cases from the GE-625/635 analysis. Each is marked to give an indication of the probable impact as we have found it (** for the most severe effect, * for the least).

3.1. Waste

Poor Code (*). Although very obvious, it is

often unimportant and may be negligible in the presence of other efficiency factors. Furthermore, poor code is the hardest kind of inefficiency to eliminate from a working program. Look elsewhere first.

*Unnecessary Actions (**).* Here are included unnecessary typeouts, printouts, etc. Corrective actions like rereading a tape block do not have to be signalled to the operator. A better method is to set a threshold value, informing the operator of a probable service need when a unit has operated in excess of that threshold. Similarly, a store dump should not be printed automatically on every program failure; the dump process may represent more work to the computer than the actual running of the program, and the dump is often useless and distracting to the programmer.

3.2. Indecision

*Manner of Waiting (**).* This concerns the manner in which one process waits for the completion of another. Two examples from a multiprogramming system are:

- When a program requires I/O and the channel it needs is busy, it should be removed from processing and put into an I/O queue; it should not repeatedly receive and then surrender control (because it cannot do anything about it) until the channel is free.
- When a non-resident module of the supervisor is required, and not enough main store is free to bring it in, the store manager should "sleep" until more free store develops, not repeatedly receive and surrender control.

*Scheduling Errors (**).* Certain efficiency errors derive from redundant scheduling of software processes. Generally, redundant scheduling occurs "just in case" the process may be needed. For example, the resource allocation function of the supervisor may be executed for every change in free resource availability, however trivial. If the resource allocator is non-resident, the efficiency degradation from secondary effects may be overwhelming.

3.3. Interferences

Interferences occur when two or more processes require the same resource.

Unnecessary Device Competition ().* Problems of this type arise primarily from design mistakes. Examples include inadequate I/O queuing provisions and lack of a centralized console output program equipped with adequate

message buffers.

*Workload Imbalance (**).* A multiprogramming supervisor should avoid creating obvious interferences such as scheduling several processor-bound programs for concurrent execution in a uni-processing environment, or scheduling two sorts on one magnetic tape channel. The user can help to balance the workload through his own operating procedures.

*Unnecessary Serialization (**).* A common intuition is that each program should minimize its resource requirements in a multiprogramming system, so that the number of concurrent programs in execution will be maximized. This idea is reversed by careful analysis. When a programmer elects non-overlapped I/O and processing, he guarantees that all of his program's resources will be tied up for the maximum possible time - in some cases 3 times as long as with full overlap. The amount of extra store required for overlapped I/O is nearly always small in comparison to the overall store requirement of the program. Basically, a program should never delay initiating anything it can usefully do. To achieve peak hardware efficiency, the supervisor requires a queue of at least one I/O request for each channel and at least one processing request for each processor. An individual program with overlapped I/O can often keep requests in two or three such queues throughout its execution.

3.4. Inadequate Resources

Given inadequate hardware resources, software cannot drive the hardware to peak performance. On the other hand, excessive resources mean excessive expense. Optimization is required.

*Temporary Store for Supervisor (**).* When a nonresident module of the supervisor is required and inadequate free store is available, either its execution must be delayed or else some other program must be swapped out. In either case performance is degraded. To minimize the consequences, the fraction of fast store reserved for non-resident modules should be tuned for each user installation, and perhaps varied to reflect cyclic changes in workload.

Secondary Store ().* For a multiprogramming system, maximum throughput assumes concurrent execution of multiple software elements. The amount of secondary store reserved for this purpose must also be tuned for optimal performance in each user installation. Similarly the total number of magnetic tape handlers and channels, the total capacity of the primary and

secondary store, and other related parameters require tuning to the actual workload.

3.5. Poor Resource Deployment

*Poor Blocking (***)*. Use of small blocks on bulk storage devices invokes extra delays for device latency, in practice by 2 to 1 or more. In a multiprogramming system, with facilities timeshared between programs, one program's poor blocking generally delays that program and others as well.

*Poor Data Organization (**)*. Two cases of data organization on rotating storage are particularly significant. First, serial searching of serially ordered files is unnecessary and quite wasteful. Second, the space allocation strategy on any movable arm device should carefully minimize arm movement.

Alternate Device Tradeoffs ()*. If given magnetic tapes, disc and high speed drum storage in a multiprogramming environment, which medium shall be used for each of the system software files and each application program file? A serially-accessed file could reside on any of the media mentioned, but limited drum space, relatively slow disc speed, and tape mounting time must be considered. Again, tuning to the user's actual workload.

3.6. High Overhead

We come at last to the universal scapegoat. By "overhead" we understand the fraction of total hardware capability required for the necessary work of the supervisor. Intuition attributes most observed inefficiency to this contributor. However, the actual degradation of system productivity resulting from overhead is likely to prove to be a low order effect after other inefficiency causes are removed. Nevertheless, sources of high overhead merit attention.

*Unit Record Handling (**)*. If print lines are issued one at a time by the software, with a processor interrupt and dispatching of a media conversion program as required for each line, the system efficiency must be much lower than with multiple line bursts. Most of the capability of the resident supervisor is involved in the former cases, while in the latter a small extension to the interrupt handler (or better yet, the I/O hardware) suffices for most print lines.

*Worst Case Orientation (**)*. For the decision logic in any program there is usually a statistically dominant combination of properties which occurs in anywhere from 50 to 90% of the transactions. A program should be organized to re-

cognize the dominant pattern in a minimum number of tests (the first to be made!) and thus avoid all other testing or special action required for the minority cases. A 2:1 efficiency ratio is often at stake.

*Expensive Elegance (***)*. Special features and degrees of freedom calculated to whet the fancy of the sophisticated programmer, yet rarely used, can degrade overwhelmingly the system productivity. The reader is invited to think of his own examples.

4. SOME HARDWARE DESIGN IMPLICATIONS

Basic facilities for software instrumentation are a rundown timer (which generates program interrupts) and an accurate time-of-day clock.

Even after software is free of major flaws, system performance still varies with workload. As presented to the system at different times, the same program may require various treatments of control conditions to run optimally. If the job mix is presented by human operators, some variables of the system may be controlled at human interaction time. Choices may be made for:

- facility assignment and usage (presuming interfaces such that facilities are interchangeable in usage)
- one of a multiplicity of algorithms for each main control function
- program introduction time and mix (operator may choose to delay or reorder programs)

Theoretically most of these adjustments could be made automatically by the software. In practice some must be relegated to operator control. Information can be supplied to the operator in simple form, such as a visual display indicating per cent loading (continuous) of the facilities - store, drum, disc, tape, etc.

A console facility should be provided for the operator to bias these assignments. The software can interrogate the dial settings periodically and update the control parameters. The operator could even try various settings experimentally, resetting previous conditions automatically in case he overshoots.

5. SOME SOFTWARE DESIGN IMPLICATIONS

Some lessons learned from this work are:

- Design software units to be as self-contained as possible, to allow easier and freer choice of alternatives and modifications

- It is mandatory not to fix the variables of a system by imbedding them implicitly in the programs
- However, it is preferable to fix the variables explicitly unless the user is supplied with advice on how to set them, and instrumentation to measure the effect of the setting.

REFERENCES

- [1] H. Sackman, W. J. Erikson and E. E. Grant, Exploratory Experimental Studies Comparing On-Line and Off-Line Programming Performance, Comm. ACM 11 (1968).
- [2] H. N. Cantrell and A. L. Ellison, Multiprogramming System Performance Measurement and Analysis, Proc. AFIPS Conference 1968, Vol. 32, pp. 213-221.

DISCUSSION

Remark by P. Samet

I entirely agree with you. There is a great need for proper accounting with all jobs. At our installation at University College, London, we have recorded the steps into which people's jobs fall. At the end of the week we automatically know how much time was spent compiling pro-

grams, executing, link editing. From these statistics we can determine which steps are worth investigating.

Answer

This method is the beginning of attacking the inefficiency. When the main source of inefficiency has been determined, it is then possible to attack the finer causes.

Reprinted from
Annual Review in
AUTOMATIC
PROGRAMMING
Volume 5

5 4

A Politico-Social History of Algol

R. W. Bemer

Edited by
Mark Halpern and Christopher Shaw

PERGAMON PRESS OXFORD - NEW YORK 1969

begin comment whatever happened to the International Algebraic Language? From a much heralded birth in the late 1950's, the language which became ALGOL 60 seems to have pined away for lack of support. Is ALGOL dying a lingering death because it doesn't have roots to grow from, or because it is being secretly strangled by a better-established language?;
end;

The most widely used algorithmic language today is FORTRAN in its several variants. It is implemented on virtually all the currently used machines and even the rawest computer novice can probably claim some acquaintance with the language. Yet FORTRAN is not an easy language to learn or apply. Since it was designed for formula translation, algebraic statements are relatively simple to express but the declaration of variables and the specification of input-output make so many concessions to the computer that ease of use is lost. One of the design goals of FORTRAN was to provide efficient object code and this, to some extent, it does but at the expense of ease of use.

In May, 1958, a group of U.S. and European computer experts met in Zurich to report on the progress of a working party on a universal language. The language adopted by this meeting became ALGOL 58 and later ALGOL 60. As a programming language, ALGOL has tremendous power and flexibility and yet is more easily comprehensible and usable by both experts and beginners. Even if you are not an expert in either language they may be compared by looking at the algorithms published in the Communications of the Association for Computing Machinery. The two official publication languages are ALGOL and FORTRAN. There can be no doubt as to which algorithms are easier to comprehend.

ALGOL has found wide acceptance in Europe but not in the U.S., except perhaps as a vehicle for the academic interchange of algorithms. In the U.S., ALGOL should have progressed rapidly. The IBM user's group (SHARE) gave it initial support and proposed that it be adopted as a SHARE standard. Later SHARE quietly dropped this proposal. The first problem with ALGOL was one of maintenance and modification. ALGOL was not supported by an international body such as the International Standards Organization, nor was it under the wing of one manufacturer as was FORTRAN. As a result it was not only difficult to get the progenitors of ALGOL 60 to modify their language but also users could not be prohibited

from changing whatever they wished. This gave rise to a number of ALGOL variants such as the University of Michigan's MAD, System Development Corp's JOVIAL, and the Soviet Union's ALPHA.

The second hurdle that ALGOL had to surmount was that of the vested interest in FORTRAN. Many users already had a considerable investment in FORTRAN programs and were not willing to reprogram for another language. This might have been overcome if the principal supporter of FORTRAN, IBM, had been prepared to support ALGOL wholeheartedly and devise not only compilers, but also FORTRAN/ALGOL converters, or allow FORTRAN routines to be called by ALGOL programs. However, IBM was never really enthusiastic about ALGOL, perhaps because IBM did not devise it. Eventually IBM came up with PL/I and assumed that ALGOL would quietly disappear.

A third problem was that most of the early published discussion of ALGOL centered on its more advanced features, leaving the impression that ALGOL was useful only for difficult mathematical programming. The proponents of ALGOL did not produce a beginner's primer which could have shown the simplicity of the language and thus obtained a wider appreciation of its virtues.

But the virtues of ALGOL still stand today. It uses ordinary mathematical notation, ordinary English words, and does not have a profusion of arbitrary restrictions and exceptions. ALGOL is a universal language and as such can be used for communication between users. For the sophisticated user it is powerful and flexible yet simple enough to be easily learned. It is also very suitable for conversational time-sharing use. In short it is a language for the computer user. ALGOL could be re-established in the U.S. if computer users would only demand it of the manufacturers. We have had enough of having the large computer companies satisfy only those customer demands that are in the manufacturers interest. Users should tell the main frame manufacturers what languages to offer.

Sam Alder

A Politico-Social History of Algol[†]

(With a Chronology in the Form of a Log Book)

R. W. BEMER

Introduction

This is an admittedly fragmentary chronicle of events in the development of the algorithmic language ALGOL. Nevertheless, it seems pertinent, while we await the advent of a technical and conceptual history, to outline the matrix of forces which shaped that history in a political and social sense. Perhaps the author's role is only that of recorder of visible events, rather than the complex interplay of ideas which have made ALGOL the force it is in the computational world. It is true, as Professor Ershov stated in his review of a draft of the present work, that "the reading of this history, rich in curious details, nevertheless does not enable the beginner to understand why ALGOL, with a history that would seem more disappointing than triumphant, changed the face of current programming". I can only state that the time scale and my own lesser competence do not allow the tracing of conceptual development in requisite detail. Books are sure to follow in this area, particularly one by Knuth.

A further defect in the present work is the relatively lesser availability of European input to the log, although I could claim better access than many in the U.S.A. This is regrettable in view of the relatively stronger support given to ALGOL in Europe. Perhaps this calmer acceptance had the effect of reducing the number of significant entries for a log such as this.

Following a brief view of the pattern of events come the entries of the chronology, or log, numbered for reference in the text. These log entries are taken primarily from published articles, news and minutes. However, they are necessarily much abridged, as people seldom write compactly for history. The responsibility for their choice and abridgement is mine, and considerable care has been taken not to quote out of context and either misrepresent or misportray any of the actors (mostly

[†] The introductory text is adapted from the introduction to C. P. Lecht, *The Programmer's ALGOL*, McGraw-Hill, 1967.

living) in the ALGOL drama. Also included is a reasonably comprehensive bibliography of papers, books and meetings which show primary emphasis on ALGOL.

The Pattern

ALGOL was started with high hopes for both universalism and efficacy. The first occurred slowly because of business practicality, the slowness of communication in such a large field, and the lack of a controlling body. The second lagged because such a language matures slowly, being dependent upon actual usage and experience for feedback and improvement. Some of the U.S. participants in defining ALGOL 60 had just returned from the Paris meeting when a so-called "rump" group demonstrated the ambiguities they had unwittingly approved.

One can give several reasons for this slow maturation. They are:

1. INEFFECTIVE DESCRIPTION AND PUBLIC RELATIONS

The advantages and power of ALGOL (perhaps as compared to FORTRAN) were not obvious to all because the desirable nature of its improved logical rigor and generality were not publicized effectively [75]. The first, and perhaps the fortieth, description of the language was couched in such terms as to repel the practicing programmer of that day, particularly in the U.S. That this trepidation was due to the form of presentation was illustrated strikingly by Rabinowitz's paper, "Report on the Algorithmic Language FORTRAN II," in the 62 June issue of the *Communications of the ACM*. The title is identical to that of the Paris report except for the name of the language, and the description is given correspondingly in Backus Normal Form. As such, it looks far more forbidding than ALGOL; at the very least it demonstrates many more exceptions and structural faults.

Supporters of the language were in agreement that action was needed [77], and some arguments were made that usage was not really that difficult [107]. More than this, correct ALGOL usage seemed easier to achieve [115]. Other attempts were made to popularize the representation [75], but the flexibility and power obviously frightened the average practitioner.

2. DIFFERENCE IN ORIENTATION BETWEEN THE U.S. AND EUROPE

In 1958 the volume of work done with a formula language in Europe was only a fraction of that in the U.S., roughly paralleling the disparity in numbers of computers in use. Thus a reasonably fresh start was

possible [2]. The U.S. community felt itself to be more practical and suspected the Europeans of idealism. Indeed, it was not possible to assume that professional adoption of ALGOL in Europe implied full international acceptance [87]. By number of countries, yes; by number of users, no. An ISO standard for the language is possible only at the end of 1967, because requirements for such a standard transcend what was envisioned in 1958, or even 1961.

① ALGOL should have progressed more rapidly in the U.S., for the powerful SHARE organization certainly gave it initial support [14]. SHARE had planned to stop further modification to FORTRAN and adopt ALGOL [19, 26]. Yet later it withdrew acceptance [93] and proceeded with FORTRAN IV, even though that language was also incompatible with its predecessor [74]. Primarily this was due to a vested interest in FORTRAN programs [112, 128, 138], despite published reasons of failure to achieve a successful processor [92, 93]. Europeans who puzzled over this, in light of their own successes [36, 50], will now find this easier to understand with the recent failure of the Decimal ASCII card code (adopted for a while as an ECMA standard) to hold under the onslaught of the vested interest in Hollerith-based codes. ⑤

Whereas the U.S. Government gave strong support to the COBOL language, for business data processing, they did not to ALGOL. In contrast, the German Research Council desired that all computers at German universities be equipped with ALGOL processors as a condition of placing an order. Since it reportedly provides 95% of the funding for this purpose, the support is assumed to be strong [179].

3. ACCENT ON PRODUCTION

ALGOL came on the scene just when U.S. users were engaged in a struggle to achieve production to justify all that expensive computer equipment they had ordered for purposes of advertising and keeping up with the Joneses. Thus most ALGOL processors were experimental at a time when FORTRAN was well into production. This pushed FORTRAN from version I to II, where programs already converted into machine language could be called for execution by a FORTRAN program. In many cases these pre-compiled subprograms may have been written in another procedure language, such as COBOL. Some installations made a practice of using FORTRAN as a linkage skeleton, with the main part of the working program written in other languages. Such usage was not applied in volume to ALGOL, and some claimed [136] that it could not be done [141, 171]. Spuriously, of course (see ALGOL for the UNIVAC 1107, calling FORTRAN and machine language subprograms).

4. INEFFECTIVE MAINTENANCE

ALGOL was the first language attempted originally on an international scale. FORTRAN was for some time under the control of a single manufacturer and COBOL was under firm industry supervision. (3) ALGOL, however, was opened to all who had an interest [55]. This led to heterogenous membership in maintenance groups [84], many of whose members had no conception of what the term "maintenance" meant to the others [60]. This led to the abortive Oak Ridge proposal [64, 73]. Universities and research centers did not have the same requirements as a manufacturer, who could be forced to pay heavy contractual penalties if software did not perform properly. In this case it became "properly according to whom?" [100].

This was the problem facing IBM, whose recognized leadership in the computer field might have put ALGOL over sooner, had they been so inclined. IBM ALGOL was available [103], consisting of ALGOL where ALGOL and FORTRAN did the same function, and FORTRAN where ALGOL was lacking, such as in input-output statements at that time. But formal control was lacking, and IBM seemed to avoid wisely the political situation. While they could have, they did not wish to usurp control. Better to be prudent and not risk the vagaries of someone "actively engaged in writing programs in the ALGOL 60 language" [55] who has a committee vote equivalent to that of IBM!

This was finally expressed with sufficient strength [90, 91, 108] and control was vested [118] in IFIPS (now IFIP), a responsible body which has mounted an excellent effort leading to resurgence in ALGOL.

5. RESTRICTIONS ON CHARACTER SET AVAILABLE

The expanded character set of ALGOL (116 symbols requiring some graphic representation, such as ? for *if*) appeared far in advance of hardware which could handle it properly. The obvious device of a publication and reference language did not suffice in actuality. The ALCOR group felt this problem strongly [36, 186]. Predictions were ignored [5] (*CACM* 2, No. 9, p. 19) and then the users found out the difficulties. Meanwhile the development of the ISO (ASCII) code was taking place. Double case alphabets are still rare, although Teletype had an experimental terminal in 1965 and showed a special Model 37 terminal in Spring of 1967. A proposal was made [125] to reserve a set of switching characters, following the ESCape character, for programming language usage, but this was never understood or followed up.

6. LACK OF INPUT-OUTPUT PROVISIONS

This has been a very real deterrent to the acceptance of ALGOL for production computation [103, 132, 133, 144, 148, 161, 163]. Fortunately some excellent work appeared at a critical time [156], which has alleviated the problem significantly [187].

The Benefits of ALGOL

1. INTEREST

ALGOL provided the first big vehicle for international discussions on a commonly developed language for the computing field, and put a lot of people to thinking, which is continuing. It was acclaimed, damned, and then treated respectfully with the growth of fuller understanding. Until the end of 1960, *Datamation* (presumably the leading U.S. periodical in the field) took very little public notice. Around the end of 1961 the interest started to run high and was maintained through 1964, gradually decreasing from that time.

2. NOTATION

Inadequacies in the ALGOL 58 Report led John Backus to propose the essentials of a new method in 1959 [24]. Woodger states that prior to this no European ALGOL specification of comparable formality either existed or was considered necessary. Dubbed BNF, or Backus Normal Form, it was one of several independent developments. Knuth suggested (*CACM* 7, No. 12) that the initials stand for Backus Naur Form, but then Ingerman discovered (*CACM* 10, No. 3) virtually the same scheme prior to 200 B.C.! Regardless of this, it was the metalinguistic tool which provided impetus to further developments in languages, construction languages and processors. Many consider it the most important characteristic of ALGOL 60.

Knuth also points out how the Report has given implicitly standard terms and definitions for programming terminology, e.g., statement, declaration, type, label, primary, block, etc.

3. UNDERSTANDABLE ALGORITHMS FOR HUMAN MACHINE INTERCHANGE

The interest of most numerical analysts has been captured by the possibility of describing their processes in a way that is at the same time very readable for understanding and also suitable for machine translation on a variety of equipment to produce working programs

which perform those processes. A relatively large number of algorithms have been published in:

Communications of the ACM (U.S.A.)
The Computer Journal/Bulletin (U.K.)
B.I.T. (Scandinavia)
Algorytmy (Poland)
Numerische Mathematik (Germany)

Also in *Selected Numerical Methods* (Gram, Regnecentralen, Copenhagen, 1962, 308 pp.)
Computer Programs for Physical Chemistry
 (Maine and Seawright)

Ageev has edited a Russian translation of the *CACM Algorithms*, Nos. 1-50 in 66 May, Nos. 51-100 in 66 June. Indexes of algorithms have been published in the *Communications of the ACM*, in the issues for 62 Jan, 64 Mar, 64 Dec and 65 Dec. Apparently missing is the *Taschenbuch* [35, 59], which was to be a handbook in five or more volumes published by Springer-Verlag. Designed to be a compendium of numerical knowledge encapsulated in the ALGOL language, this is not yet available, although typescript for a part of Vol. 1 was reported to be in draft form in 66 Oct. Possibly a sufficient body of algorithms has not yet been accumulated in a comprehensive manner.

4. AS A CONSTRUCTION LANGUAGE AND FOR SUPERSTRUCTURES

I stated in 1962, at the Armour Research Foundation, that "languages of the future, whether or not they be outgrowths, modifications or adaptations of our present languages, will survive only on the basis of being both introspective and reproductive. They must have the facility to talk about themselves and specify their processor in their own language." ALGOL has been more successful than most in this area. van der Poel offered at the Rome meeting to supply a version of ALGOL written essentially in ALGOL. Burroughs did ALGOL for the B5000 in this manner, and so did Bull General Electric for the 600. More than this, it has been used as a construction language for other processors.

Both FORTRAN and ALGOL have been used as bases for superstructures of other programs, both applications and other languages. An exceptional example of this is SIMULA, done by the Norwegian Computing Centre, which is both connective to and written in ALGOL. It is my opinion that ALGOL has so far lent itself more suitably for this purpose than any competitor of standing.

5. EFFECT UPON COMPUTER DESIGN

It has been tempting to ascribe to ALGOL influence the pushdown stores of the English Electric KDF9 and the Burroughs B5000, which were introduced during the early ALGOL period, especially since the design and advertising goals of the latter machine were oriented to direct usage of ALGOL (and COBOL). This was because of translation methods variously labelled stacks, cellar, LIFO (last in, first out), and the like, which were utilized in ALGOL processors. Although these techniques are related, the basic hardware ideas were of a much earlier vintage. It is said that they appear in the B5000 because it was easier to translate to Polish notation than to one-address machine language. Direct ALGOL effects upon the B5000 are illustrated by the "operand call", specifically tailored to the "call by name" of ALGOL. Like languages also had a considerable effect upon design of computers for operating directly from source language without translation, such as the ADAM computer by Rice and Mullery of IBM.

6. A LANGUAGE FOR NEW GENERATIONS OF COMPUTERS

ALGOL X and Y are being developed as successors [166, 169, 173] to ALGOL 60. There was formerly an inexpressible and intuitive feeling by ALGOL proponents that the elegant and simple structure was of great value [122], but this could not be shown to enough advantage to convince FORTRAN users. Multiprocessing, multiprogramming, reactive operation, time-sharing and realtime environments provided the crucial evaluation. The basic power of ALGOL is more evident now that facilities must be provided in a language to handle these new complications. This was evident in IBM's switch [161, 169, 172] to a new language with many of the features of ALGOL. However, IFIP has not been relaxing in its role of custodian for ALGOL. ALGOL X shows many differences from ALGOL 60. For example, Naur has proposed [171] an environment and data division. The lesson of 1959 COMTRAN has been learned.

7. DISPELLING A MYTH

At the beginning of the ALGOL effort, SHARE was promoting UNCOL, or *UNiversal Computer Oriented Language* [4], but its proponents do not seem much in evidence these days. The first APIC *Annual Review in Automatic Programming* had an article indicating that no UNCOL processors were running yet, due to the fact that language specifications were incomplete. One wonders where it is after these eight years; apparently the last published paper was that in the 62 Jan

Computer Bulletin. We should hesitate to compare it to the philosopher's stone, however, because successful processors have been constructed with special purpose languages just one step up from the UNCOL concept.

Summation

A few quantitative measurements are perhaps useful to round out this special history:

1. Interest in ALGOL has been international since its inception. However, the circle of interested countries has expanded recently. The mailing list for *ALGOL Bulletin* No. 19 included recipients in:

Australia	Czechoslovakia	Italy	Sweden
Austria	Denmark	Japan	Switzerland
Belgium	France	Netherlands	United Kingdom
Canada	Germany W.	Norway	United States
China	Germany E.	Poland	U.S.S.R.

By the time of *ALGOL Bulletin* No. 25, this group was augmented by:

Argentina	Hungary	Israel	South Africa
Brazil	India	Mexico	Spain
Finland	Irish Republic	Rumania	

This same *ALGOL Bulletin* is the best way to note the progress of ALGOL X and Y. Copies may be obtained from three sources:

- (a) The Mathematical Centre, 2e Boerhaavestraat 49, Amsterdam-0, the Netherlands (attention M. F. Calisch).
- (b) P. Z. Ingerman, RCA, EDP Bldg. 204-2, Camden, New Jersey 08101, U.S.A.
- (c) SIGPLAN Notices, C. J. Shaw, Editor, System Development Corporation, 2500 Colorado Avenue, Santa Monica, California 90406, U.S.A. (The *ALGOL Bulletin* is reprinted in these notices upon issuance.)

④ 2. ALGOL is sometimes considered to have fathered a number of variants [64] such as Mad [75], JOVIAL [99, 129], CPL [167], NELIAC [129], BALGOL [115], ALPHA [182], etc., whereas FORTRAN is often considered to be pure. About the only difference I can see is that the authors of the ALGOL variants gave them different names, while the authors of the FORTRAN variants for the most part retained the name, regardless of difference in available features and operational effect of common features. One survey of the different FORTRANs (prior to standardization by ASA, now USASI) showed over a dozen, of which eight existed within IBM itself. Then too, FORTRAN II is

quite different from FORTRAN I, and FORTRAN IV goes so far as to be intentionally incompatible with FORTRAN II [74]. This is reflected in the USASI Vocabulary, which defines FORTRAN as "... Any of several specific procedure oriented languages."

3. In comparing ALGOL to FORTRAN we note the following:

(a) From a publication and paper viewpoint, the KWIC index to the AFIPS Conferences, 1951-1964, shows two papers on ALGOL and one on FORTRAN (the original), and is inconclusive. The KWIC index to Computing Reviews (ACM), 1960-1963, shows 49 papers on ALGOL to 11 on FORTRAN, but this is certainly equalized by the fact that FORTRAN is by far the earlier language. It arrived at a time when most of the present journals in information processing, such as the *Communications of the ACM*, were non-existent, and naturally the most papers would arise during the earliest life of a programming language.

(b) From the standpoint of the number of published algorithms, ALGOL holds a commanding lead.

(c) The number of books and texts could be considered roughly equal.

(d) When last surveyed, the number of processors for various computers was about equal (*CACM*, 63 Mar). Despite a formal request through the *ALGOL Bulletin*, the ISO survey [125] has not been updated in this area. W. McClelland, director of the ISO/TC97/SC5 survey at the time of its publication, reports that lack of information forced the disbandment of that subgroup. However, the number of ALGOL processors has certainly increased considerably since that time, possibly more in proportion than FORTRAN. The 64 July *ALGOL Bulletin* reports eight compilers in use in Japan, with four more under construction, where the original survey showed none.

(e) The comparative numbers of users can only be estimated, based upon such information as [185], which showed FORTRAN programs at about ten times the number of ALGOL programs (for the U.S. only), but one could guess that perhaps only four times as many FORTRAN programmers exist, which seems quite remarkable in view of the previous quantitative comparisons. Even this comparison can be faulty, for it does not consider the increasing proportion of programmers who know and utilize both languages.

Concluding, I commend ALGOL and its future to the independent thinkers like Professor Galler [178]. If something proves practical and of substance, use it, but not for the sake of nationalism, entrenchment or prejudice. ALGOL, in its many manifestations and effects, has won a secure place in information processing history.

The Algol Log

- [1] 57 Resolution signed by twelve representatives of various
 May user groups such as SHARE, USE and DUO, at a meeting
 10 in Los Angeles:

"We, as users of diverse machines, recognizing that developments in the use of automatic computers are leading to techniques of programming which transcend the characteristics of particular machines, that communication between users of different machines is highly desirable, and further, that completed programs which are machine independent appear to be possible, recommend that the ACM take the following action:

- a. Appoint a committee to study and recommend action toward a universal programming language.
- b. Set up means for the rapid exchange of practical information on computer programs and programming among all computer users.
- c. Appoint a committee to study and recommend areas of standardization. . . ."

(Reported in DATA-LINK, 57 Oct)

- [2] 57 Letter from GAMM members to Prof. John Carr, III,
 Oct President of ACM:
 19

"We, that is, Messrs. Bauer, Bottenbruch, Lauchli, Penzlin, Rutishauser and Samelson—have gathered here at Lugano for one of our regularly scheduled working sessions under the auspices of our formula-translation project. As Bauer and Bottenbruch have already told you in Ann Arbor, we are working on the logical structure of a formula translator that will form the basis for the formula translation program of the computational groups in Darmstadt, Munich and Zurich, and in the future at several other European installations. The program committee of the *Gesellschaft für angewandte Mathematik* (GAMM—Society for Applied Mathematics) under the chairmanship of Prof. Heinhold is functioning as the coordinating agency. Considering its circumference and the direction in which the project has moved, it appeared to us from the beginning that only a joint effort was possible despite the underlying difference of the machine types involved.

Our work is largely finished. In particular we have, after overall discussions, agreed on one language that in our opinion fulfils the following basic conditions:

- (1) It depends directly on ordinary mathematical formula language.
- (2) It is "self-explanatory".
- (3) It brings directly into the expression the dynamical nature of the calculational event.
- (4) It is independent of the technical characteristics of the computer.

In order to stay in the area of what we believe we can cover, we have confined ourselves to the description of "scientific computations". We have endeavored from the beginning to avoid the possibility of deviations from existing earlier proposals (Rutishauser 1951, FORTRAN, partly also PACT).

Guided by the news and reports that Bauer and Bottenbruch have brought back from America, we have decided that our hitherto existing proposals also largely agree structurally with Perlis' IT language and the Remington-Rand Math-Matic. This agreement is most striking with Math-Matic, the most recent of the listed proposals.

We consider it a misfortune that at this time several different languages exist, but none of these languages appears to overshadow the other enough so that this would offer a reason for selecting it. We would like to avoid increasing this bad situation by setting up in Middle Europe one more such language.

Bauer and Bottenbruch have talked with several mathematicians, in particular also with you, about these questions. From this the idea here has gradually crystallized from a joint conference to make the attempt to work out a basis for a uniform formula language. However, this must be set about at once, since in the present state of development in a few months it will already be definitely too late when the different languages will not only be installed in the different user circles but in use.

We would therefore make the following proposal to you as the President of the ACM: The President of the ACM and the President of the Programming Committee of the GAMM issue a joint call for a closed conference of those people active in the area of formula translation. The task of this conference shall be:

- (1) To clarify how much the logical structure of those formula languages which are already in existence permit an adjustment,
- (2) To fix upon a common formula language in the form of a recommendation.

... Bottenbruch and Bauer have informally already talked with persons at IBM and Remington-Rand who have stated their interest. We think, however, that also the Universities should be represented in order to be able to contribute the experience of users. We hope to expand the circle through representatives from England, Holland and Sweden. . . .

For dates we would propose January-February 1958, length 2-4 days. . . .

We are presently preparing a comparative summary of the existing completely constructed formula languages, which could be placed at the disposal of the participants as the basis of the discussion. . . ."

- [3] 57 John Carr, President of ACM, to the ACM Council:
Oct "I am enclosing a letter recently received from Drs. Bauer and
26 Samelson of the University of Munich, Dr. Bottenbruch of Technische Hochschule, Darmstadt, and Prof. Rutishauser and Drs.

Lauchli and Penzlin of Zurich, Eidgenossische Technische Hochschule.

The letter is generally self-explanatory. In light of the resolution of the National Council of the ACM in June at Houston, I am tentatively accepting the invitation of the European group to hold a meeting on a universal computer language during the period from about January 20 to February 7, 1958,

Composition of Delegation. I am also proposing that there be an American delegation of six persons to this conference, three from industrial organizations and three from Universities . . . one individual each from Professor Morse's laboratory at M.I.T. and from Professor Perlis' laboratory at Carnegie Tech, and myself, representing the University of Michigan and the A.C.M. . . ."

- [4] 57 Francis V. Wagner, Chairman of SHARE, to SHARE
Nov Executive Board:
22 ". . . I believe very, very firmly that the establishment of a universal algebraic language for programmers to code in is a relatively trivial project. I do not feel that the existence of several such 'higher order' languages would particularly hurt the computing profession. (In fact, I think it necessary that there be many, each adapted to its own field.) On the other hand, I am absolutely convinced that the most important thing that is needed is a universal, intermediate, 'computer' language as described by Charlie Swift. . . .
I propose that we urge this august, academic body convened in Switzerland to not waste their time with universal algebraic 'programming' languages, but to devote their efforts exclusively to the important matter of an intermediate universal 'computer' language for a universal pseudo-computer. . . ."
- [5] 57 H. S. Bright to Professor John W. Carr, III:
Nov ". . . Although this might be difficult on a world-wide scale, I believe
26 that early profit could be achieved by some standardization on language elements at the first level above the bit, viz., at that of alphabet or, more generally, of characters commonly written as the least unit of language. . . ."
- [6] 57 Francis V. Wagner, Chairman of SHARE, to Dr. John
Dec W. Carr, III:
9 ". . . We are pleased that, as President of A.C.M., you are coordinating the affair and establishing the ground rules for the selection on the United States Delegation. We think, however, you are making a mistake in loading it so heavily with compiler designers and university people. . . ."
- [7] 57 John W. Carr, III, to ACM Council:
Dec ". . . I have also talked briefly with Professor Perlis of Carnegie
13 Tech, who indicated plans for a possible meeting among interested Americans at the Eastern J.C.C. meeting in Washington in December. . . ."

My personal feeling is that a great deal can come out of such a conference—not necessarily a common language, which I doubt can be achieved at one fell swoop, but rather an overall plan for arranging to translate languages one into another, standards for such languages, and at least a meeting of the minds on the goals and ways of reaching them. On this basis I feel that this, along with the contacts with the European group, could be of great benefit. . . .”

- [8] 57 Francis V. Wagner to Dr. John W. Carr, III:
Dec
20 “... It seems to me a shame to waste all this time and effort on just another algebraic higher order language even though it purports to be ‘universal’. It seems to me that such an assumption is almost a contradiction in terms . . . the most useful manner of exploiting the computers of the future will be to encourage every discipline to develop a higher order programmer language which most ideally suits its subject matter. Thus there should be programmer languages for aerodynamicists, petroleum engineers, nuclear physicists, medical diagnosticians, clothing manufacturers, etc. Even if this were not technically sound. . . . I maintain that human nature will make it inevitable. Thus an algebraic programmer language can never be universal, for lack of universal acceptance. . . .”
- [9] 58 ACM Ad Hoc Committee on a Common Algebraic Lan-
Jan guage, first meeting.
24
- [10] 58 XTRAN Announcement to SHARE, being an experimen-
Feb tal language intended to have the capability to express its
26 own processor.
- [11] 58 ACM Ad Hoc Committee on a Common Algebraic Lan-
Apr guage, third meeting.
18 Professor Bauer was present and described the GAMM proposal. The ACM “Proposal for a Programming language”, 19 pp. in ditto, was prepared.
- [12] 58 ALGOL Meeting in Zurich, attended by:
May
27 GAMM—Bauer, F. L. ACM—Backus, J. W.
Jun Bottenbruch, H. Katz, C.
2 Rutishauser, H. Perlis, A. J.
Samelson, K. Wegstein, J. H.

The result was a report prepared by Perlis and Samelson, published naming the language both IAL (International Algebraic Language) and ALGOL (in later publications). Often called ALGOL 58 as a means of distinguishing it from ALGOL 60, although purists frown upon this.

- [13] 58 John Backus, SHARE Representative to Zurich Meeting,
Aug to SHARE:
14

"... It seems to me that this report represents a considerable step forward for that part of the scientific community interested in numerical computation. . . . It already appears that the language proposed will be used widely throughout the Continent (work on translators for a number of European machines has been started) and very likely in this country. . . .

In conclusion, as your SHARE representative on the ACM Ad Hoc Committee on Languages, I want to urge SHARE to consider giving official recognition to the language proposed here. I do so because I am convinced that it is fundamentally sound, that no better language is likely to be approved in the near future by an international group representing the outstanding computing societies of the United States and the Continent, that the goals of SHARE would be greatly advanced by recognizing and using it, and finally that SHARE, by its adoption, would be making a major contribution toward transforming the field of numerical computation from a somewhat parochial and divided enterprise into a truly international scientific discipline."

- [14] 58 Resolution adopted by SHARE XI:
Sep
11

"SHARE by this resolution commends and endorses the work of the ACM-GAMM International Conference on Algebraic Language, and in particular of SHARE's representative, J. Backus, and his American colleagues, C. Katz, A. Perlis and J. Wegstein.

Furthermore, SHARE intends to use this language as soon as it can be implemented. To this end, SHARE will take positive action to study the proposed International Algebraic Language and to implement its adoption as a SHARE standard.

The immediate work should be carried forward by an Ad Hoc Committee on Algebraic Languages, which the President of SHARE is directed to appoint on September 12."

(Frank Engel was appointed chairman.)

- [15] 58 Report by A. E. Glennie at SHARE XI:
Sep
12

"Although British representatives did not attend the ACM-GAMM meeting on IAL, this does not mean lack of interest in the subject. In fact there have been at least six automatic coding systems used in England, the first in late 1951.

The growth of automatic coding has come recently only with the advent in England of machines with large core storage systems. Earlier machines had storage mainly on drums, which makes automatic storage allocation extremely difficult.

The British Computer Society is now awaiting the IAL proposals with a view to recommend them as standards for future British work in this area."

- [16] 58 J. H. Wegstein letter to the ACM Council:
Oct " . . . Now that SHARE is supporting it, the IAL now appears to
20 have an excellent chance for success. . . ."
- [17] 58 Informal Meeting of the European ALGOL group at the
Nov University at Mainz.
- [18] 58 Minutes of the ACM Council:
Dec "The following resolution was adopted:
3 "The Council commends Professor Perlis on the activities of his
Committee and urges him to do everything possible at the Inter-
national Conference in Information Processing in order to secure
the international adoption of a universal computer language. The
Committee is further urged to work closely with the various User
Groups to secure domestic adoption also." "
- [19] 59 IAL Committee Resolutions at SHARE XII:
Feb "*Resolution No. 1*
18-20 Whereas SHARE recognizes the importance of maintaining, at any
moment, a precise definition of the IAL (International Algebraic
Language) which constitutes in every detail an official ACM and/
or International Standard; and whereas SHARE also recognizes that
corrections, additions and improvements in IAL will occasionally
be desirable, be it hereby resolved that:
SHARE directs the executive board to take whatever steps it deems
appropriate strongly to encourage the ACM to establish formal
machinery for considering and giving official status to alterations in
IAL as a computing standard.

Resolution No. 2

The SHARE IAL Committee after extended discussion has agreed that an extended character set will eventually be required, and that for the effective implementation of the IAL language an extended set of at least 100 characters is needed now. We propose the following resolution for consideration by the SHARE body:

Whereas we deplore the inadequacy of the presently available limited character set and feel that more than 128 characters would be desirable,

be it resolved that SHARE recognizes a growing need for a more extensive character set, and recommends that IBM consider providing across-the-board input/output equipment to meet this need.

Resolution No. 3

Whereas SHARE considers that the IAL language should become a working language for communication with the 704, 709 and other SHARE machines, therefore be it resolved that:

In order to implement the creation of a working language, SHARE recommends that IBM begin development of an IAL translator; and

- [25] 59 Letter from R. W. Hamming (President, ACM) to M. A.
Aug Danjon (President, Association Française de Calcul) and
5 M. V. Wilkes (President, British Computer Society), in-
viting participation in the international ALGOL work.
An enclosure, signed by Hamming and Sauer (President, GAMM)
in Paris during the June ICIP. Without dateline, address or other
heading, it states:
“... The existing ACM-GAMM committee considers itself now
a steering committee, whose responsibilities are:
(1) to complete the Zurich report with respect to inconsistencies
and obvious extensions (e.g., the Heise committee report);
(2) to determine the procedure by which the membership will be
modified from the pool of representatives specified by the
various national (and supranational) organizations.
The selection procedure for membership in an international ALGOL
committee will always be determined by competence and no fixed
apportionment of members by nationality or organization will be
considered.”
AFCal did not accept the invitation, B.C.S. did (59 Dec).
- [26] 59 SHARE ALGOL Committee Meeting, Seattle:
Aug “The following motion was passed unanimously:
17-18 ‘Whereas an orderly curtailment of modifications in FORTRAN is
in process, looking toward replacement by ALGOL, motion No. 3
of SHARE XI is redundant and will not be resubmitted by the
ALGOL committee.’
Motion No. 3 had to do with fixing a date for ending FORTRAN
modifications.”
- [27] 59 Minutes of SHARE XIII, Seattle, reporting on a compari-
Aug son between ALGOL and FORTRAN, by Bill Heising of
19 IBM:
“1. ALGOL provides a media for universally describing problem
procedures since it is not tied to a particular machine.
2. It is expected that ALGOL will be translated for a large variety
of machines. Thus problems will not have to be recoded for
various machines as in the past.
3. ALGOL is a better and richer language than any existing to
date. . . .”
- [28] 59 Extracts from the Minutes of SHARE XIII, Reports on
Aug the ICIP Conference in Paris:
19-21 Frank Verzuh—“later, Tom Steel will inform you about ALGOL—
a language I consider important, very important to you people. . . .
Again, rather hurriedly, in Denmark, I enjoyed seeing what was
being done by the Danish Institute of Computing Machinery, the

DASK Computer, and their application of it. I was very amazed, wherever I went in Europe, people would talk to me in ALGOL language, write it and describe their work on converters, translators, etc., which are being used."

Tom Steel—"The ALGOL performance at ICIP was quite interesting in several ways. There was a section in the plenary section devoted almost entirely to ALGOL. It was billed as an automatic programming discussion, and it tended to be about very little other than ALGOL. . . .

Through this whole series of discussions there was a sort of running tacit assumption that ALGOL was a good thing. It seems there were some folks there that didn't believe this, and a gentleman named Strachey from the United Kingdom got up and challenged this assumption publicly, and he challenged all comers to a debate. This debate actually took place. It was a special rump session of the conference and one entire morning was devoted to this. There were probably eighty or so people that participated in this discussion and it got rather heated a couple of times; in particular, there was one case when one individual made a comment, and somebody down the room made a snide comment about the competence of certain people, and unfortunately, his mike was live. Well, the net result of these discussions was really two-fold; one, I believe a recognition on the part of some of the proponents of ALGOL, particularly the Europeans who were relatively unfamiliar with data processing problems, that ALGOL is not (in its present form at least) a completely general programming language, that it is not satisfactory for certain types of data processing work. I believe nearly everybody recognizes this, but the way the discussion was going, it appeared that this whole problem was being glossed over, and this debate was well worth it, if this situation was cleared up, and I believe it is fair to say it was.

As an outgrowth of it, a group of people, including Samuelson and Bauer, people from the United States, got together and decided on a way of proceeding to recommend extensions to ALGOL that will handle the data processing area. There will be a meeting of this group in conjunction with the ACM-ALGOL committee at the next ACM meeting, next month, and I suggest that any of you who feel strongly about this subject and are interested, get in touch with the proper people, Bob Bemer of IBM, Bob Bosak of the System Development Corporation, and make your ideas known and participate in this meeting.

Actually, the debate as such ended with no conclusions drawn, each side was just as sure it was right as before, as one might expect. But it did clear the air a bit and explain to many people the differences in point of view. In particular, the British seemed to feel that now is too early to adopt such a language and it doesn't look like their mind is going to be changed by talk.

In one of the symposia, there was a certain amount of discussion regarding the implementation of ALGOL, particularly in Europe. There are about five different efforts going on, largely in Scandinavia

and in Germany, where the processors or translators from ALGOL to proper machines are being conducted.

However, these processors are really not very ambitious. A rather difficult problem of procedures is just being glossed over at the moment. The translators are designed to do little more than scan single statements and construct arithmetic sequences. A great deal of effort is being devoted to such things as minimizing the number of temporary storages required for arithmetic sequence and this type of thing.

When we first heard about this, it was a little surprising. It seemed like a pretty low-level start. However, most of this work is being done on small machines with the main memory being a drum memory, and this clearly complicates enormously the problem of writing a general translator.

The Europeans were quite interested in the activity that is going on in this country toward implementing ALGOL, and a number of ideas were exchanged that, I think, will ultimately prove quite fruitful."

- [29] 59 Aug 21 Resolution adopted at SHARE XIII, Seattle, for general distribution:

"SHARE has already commended the ACM for its sponsorship of the American effort in the design of the prototype version of ALGOL. However, it deplores that current work on its development and implementation is receiving no leadership from ACM, so that user groups and independent organizations must provide their own, and coordinate only on a haphazard basis. . . ."

- [30] 59 Nov 5-6 ACM Programming Language Committee meeting in Washington:

"... The purpose of this meeting (at the National Bureau of Standards) is to discuss:

- (i) The resolution of ambiguities in (the first draft version of) ALGOL.
- (ii) The drafting of a set of improvements and extensions to ALGOL is recommended by some of the ACM membership through proposals published in the *Communications* during 1959. . . .
- (iii) The selection of a subcommittee to represent the ACM at the second international conference on ALGOL to be held somewhere in Europe early in January, 1960. . . ."

A. J. Perlis, Chairman

Membership: Backus, Green, Katz, McCarthy, Perlis, Turanski and Wegstein (for Paris), Bemer, Evans, Gorn, Rich, Dobbs, Desilets, Goodman and Levine.

- [31] 59 European ALGOL Conference at Compagnie des Machines Bull, Paris:
Nov 12-14 "Subject—Discussion of the proposed modifications of ALGOL, and preparation of the International ALGOL Conference (to be held in the U.S., possibly Philadelphia, around New-Year. . . ." (49 attendees.)
- [32] 59 ACM Programming Language committee meeting in
Nov Boston, in preparation for ALGOL 60.
30
- [33] 59 A. W. Holt to A. J. Perlis and the ACM Programming
Dec Language Committee:
1
"At your invitation, I came to Washington D.C. on November 6, 1959, in order to present to that committee a description of "Canonical Form" for programming languages—one of the results of the work of W. J. Turanski and myself. . . .
With reference to ALGOL (in its present state) there are several features of that language which concern themselves with canonical form functions (such as DO statements). In fact some of these features lie at the base of current controversy within the Language Committee precisely, I believe, because there has not yet been recognized a clear-cut distinction between signals which serve copy-edit functions vs. signals which ultimately refer to the flow of control."
- [34] 59 From the Minutes of the December 2 Meeting of the
Dec String and/or Symbol Manipulating Subcommittee of
2 ACM Programming Languages Committee:
. . . The main subject of the meeting was the string manipulation facilities to be added to ALGOL. The only recorded agreement was that the strings in question are strings of ALGOL characters. . . .
- [35] 59 Minutes of the ACM Editorial Board Meeting:
Dec
2 "The Communications, under the editorship of Joe Wegstein, will start a new department for algorithms in ALGOL language. This is a venture analogous to that proposed by Springer-Verlag (for which A. S. Householder is an editor). Our algorithms may be published eventually in the Springer Handbook; in any event, we look for a reciprocal relationship with that firm. . . ."
- [36] 59 ALCOR Hardware Group Meeting in Mainz, Germany,
Dec at the Institut für Angewandte Mathematik.
17-19

- [37] 59 Professor F. L. Bauer to the Conference members:
Dec
19 "In accordance with existing agreements, you are cordially invited to take part in the International ALGOL Conference 1960, to be held in Paris, beginning January 11, 1960, at 9:30 a.m. Participants will meet at IBM World Trade Europe. . . . The purpose of this meeting is to produce the ALGOL 1960 report based on the material screened by the American and European groups."
- [38] 60 Meeting of the "Paris 13" to produce the ALGOL 60
Jan Report.
11-16 Originally 14 members existed, but William Turanski of the ACM delegation was killed in an accident just prior to the meeting. The report is dedicated to his memory.
- [39] 60 Ascher Opler, in *Datamation*:
Jan "The transition from acceptance of FORTRAN to acceptance of ALGOL must take place in the next couple of years. . . ."
- [40] 60 Julien Green of IBM lectures at Johannes Gutenberg
Jan University, Mainz, on: "Processing of the formula lan-
19 guage ALGOL."
- [41] 60 M. Woodger to P. Naur and K. Samelson:
Jan
25 "I enclose a syntax of ALGOL 60 which is complete in as far as I understand the agreements reached on Saturday, 16th January in Paris . . ."
It appears that at least partial credit for editorial work on the ALGOL 60 Report must be extended to Mr. Woodger as well.
- [42] 60 Peter Naur to the members of the ALGOL 60 Committee:
Feb
4 "Enclosed I send you the first draft of our report. . . ."
- [43] 60 ALGOL Committee Report, SHARE XIV:
Feb
17-19 "The SHARE ALGOL Committee . . . heard a discussion of the UM MAD Compiler for the 704; the relation of its language to ALGOL, and some of the features of the processor itself.
Inasmuch as UM MAD is about to be offered for distribution through SDA the Committee, after due consideration, decided to take no action relative to accepting UM MAD as the SHARE 704 ALGOL Compiler because of the variance of MAD from ALGOL as adopted by SHARE. . . .
The SHARE ALGOL Committee has seen a draft of the ALGOL 60 and believes it to be a substantial improvement over the previous version. When it is published by the ACM the Committee recommends that the SHARE membership use the language for publication of procedures in order to further the development of the language.

IBM reported that final checkout of the second 709 ALGOL processor is under way. It is expected that this processor will be ready for distribution in May, 1960. This version of the processor produces SCAT instructions as output and it is intended as an interim processor to provide a further developmental experience with ALGOL."

- [44] 60 Remarks by A. L. Harmon to SHARE XIV:
 Feb ". . . Since the development of the ALGOL language has not reached
 17-19 the point where it seems advisable to expend the manpower required
 for a full processor that SHARE seems to deserve, based upon the
 recommendations of the SHARE ALGOL Committee, IBM will
 not produce an official ALGOL processor at this time. However,
 IBM will continue to support the ALGOL efforts in the areas of
 language development, translation techniques and, of course, pro-
 cessor development.

Questions and Answers

MR. FRANK ENGEL (WH): I believe as I entered the room I heard the statement to the effect that IBM is not intending to produce an ALGOL processor at this time. Is that correct?

MR. HARMON: A full ALGOL processor. That is correct.

MR. ENGEL: Oh; we're going to qualify it.

I understand that IBM is committed to the SHARE ALGOL Committee and to the SHARE body to produce an ALGOL processor operating in May of this year.

MR. JULIEN GREEN (IB): We did promise to have an experimental version ready by May. We can have this version ready. But this will not be a full-blown processor in the sense that, well, we didn't promise to have it to do sufficient coding. We are supposed to have the output in a form so that it will have to go into the SOS system before you can get your object programs, and this is as far as we have the processor at this point, and this is what we could have available, but this is merely for testing the language and for getting used to the language, rather than producing production programs, let's say.

MR. M. A. EFROYMSON (ER): I believe that some sentiment was given in past meetings that there would be an attempt at continuity of effort so that there would be a logical transition from FORTRAN to ALGOL, through some kind of processor between the two systems. I am not clear from your remarks whether this consideration of the evolution or revolution from FORTRAN to ALGOL is still the philosophy or not.

MR. HARMON: Yes, this is still our philosophy, and for further amplification of this I would like to again ask Julien Green to make some comments.

MR. GREEN: I think at one point we do want to use an ALGOL language. However, I don't think we are prepared at this time to cease all FORTRAN effort and say "Let's transfer immediately to

an ALGOL language," because I don't think the ALGOL language has been developed to the extent where it is worth doing this.

- [45] 60 Professor M. R. Shura-Bura (Chair of Computing Mathematics, Moscow State University) to Professor John W. Carr, III:
Mar
21

"The specialists working in the Soviet Union in the region of computational mathematics and programming are developing a large interest in the project for the algorithmic language 'ALGOL'.

I would be extremely grateful to you for information about the development of the project and accounts of practical application of the ideas of the project. . . ."

- [46] 60 B. Vauquois to the Authors of the ALGOL 60 Report:
Apr
7

"The AFCAL Committee (Association Française de Calcul) has asked me to organize the diffusion in France of ALGOL 60. In order to do so, it seems that the best mean would be a translation of ALGOL BULLETIN SUPPLEMENT No. 2 into French with more examples. The next issue of the periodic paper "Chiffres" could present this translation.

Before printing, Mr. GENUYS, Mme POYEN and I could go to Mainz in order to check the validity of translation and examples with Prof. Bauer and Samelson. . . ."

- [47] 60 R. S. Barton to Millard H. Perstein, Secretary of SHARE, in SHARE Secretary Distribution No. 69:
Apr
7

"In view of your interest in programming systems and problem-oriented languages, I am enclosing for your perusal a description† of the Burroughs version of ALGOL 58. This description follows closely that published in the December 1958 *Communications of the ACM*.

A translator for this language for use with the Burroughs 220 is in field test at Stanford Research Institute this week. Translation rate averages 500 machine instructions per minute. The system is designed for "load and go" operation and has facilities for debugging programming at the POL level and provision for segmentation of programs. Certain general input-output and output editing procedures are provided. The character set used is one available on standard keypunches.

Many new techniques have been utilized in this compiler and particular design emphasis has been put on the elimination of special rules and restrictions, as well as translation speed and ease of use operationally. . . ."

† *The transliteration of ALGOL to the Burroughs Algebraic Compiler Language, A guide for the mathematically trained programmer.*

- [48] 60 J. H. Wegstein to Julien Green:
 May "After studying CLIP, OMEGA, and XTRAN, I think that we fell
 12 down at Paris in not declaring strings in Algol. The enclosed proposal† is one which I would like to discuss at the Symbol Manipulation Meeting, May 20-21.
- It seems to me that it would be very desirable to extend Algol so that some of this string work could be standardized. We find this proposed notation useful for some of our data processing problems, and it would be very nice if we could code now for our hoped-for STRETCH Computer (in 1961) in this language."
- † By Wegstein, W. W. Youden and G. M. Galler.
- [49] 60 SHARE ALGOL Committee Meeting, in Pittsburgh.
 May Agreed were:
 23-25 (a) a SHARE ALGOL 60 hardware representation,
 (b) input-output procedure specifications,
 (c) a general outline of the desirable 'debugging' features that the SHARE ALGOL 60 processor should have. . . ."
- [50] 60 First ALGOL 60 processor tested on the X1 computer in
 Jun Amsterdam.
- Constructed by Dijkstra and Zonneveld, it even handled recursive procedures. In fact, all the features of ALGOL 60 except dynamic own arrays were implemented. Operational in August 60.
- [51] 60 *Input Language for a System of Automatic Programming*
 Jul published in Moscow by Ershov, Kozhukhin and Voloshin. Published in final copy in 1961 by the Siberian Section of the Academy of Sciences of the U.S.S.R.
- This work was machine-translated by the IBM Research Center, Tape No. 1785, 132 pp. This translation is of humorous interest because "input language" was translated by the program as "entrance tongue". The authors said they were surprised that the changes to ALGOL 58 to make ALGOL 60 corresponded to their point of view, and that this was striking because they had not given out any information (preliminary) on their working ALGOL 58 processor. Actually this system and language goes quite a way beyond ALGOL 60, in particular, vector and matrix notation and operations are provided for.
- [52] 60 J. Wegstein to Members of the ALGOL Working Group:
 Jul "As various people undertake to write ALGOL 60 compilers . . .
 28 logical errors are found; and necessary changes are indicated. Obviously if ALGOL 60 is to be made to work as a common language, an effective mechanism for maintaining it must be established. Some of the Paris conferees are not following up the report with ALGOL implementation or even further interest. On the other hand, Peter Naur has recently proposed some changes (see enclosed

letter) and asks the Paris 13 to endorse them. I have asked Naur to delay publication until the U.S. ALGOL Working Group can consider them on August 22.

... Professor Perlis wishes to appoint the Working Group as an official working subcommittee of his standing Committee on Computer Languages. This subcommittee ... participate in the effort to secure international agreement on interpretations and changes to ALGOL 60. ..."

- [53] 60 M. I. Bernstein (Chairman, SHARE ALGOL Committee)
Aug to Millard Perstein, in SHARE Secretary Distribution
10 No. 74:

"If the SHARE membership (or the Executive Board) decides that they do want ALGOL 60 as a SHARE Standard Programming language, it will be up to the SHARE ALGOL Committee to produce a processor—IBM has so far refused to do the job.

The SHARE ALGOL Committee is in need of volunteers—very special volunteers—ones who are willing to work and contribute a non-trivial portion of their time to producing an ALGOL 60 processor. ..."

- [54] 60 Meeting of the ACM ALGOL Maintenance Subcom-
Aug mittee, in Milwaukee.
22

- [55] 60 From the Minutes of SHARE XV:

Sep "On July 27, 1960, Professor A. J. Perlis, Chairman of the ACM
11-16 Committee on Computer Languages, asked the ALGOL working
group to organize itself as an ALGOL maintenance group to be regarded as a subcommittee of his Committee on Computer Languages. He asked that a report be prepared for the parent committee when the next meeting is held. On August 22nd the ALGOL working group met in Milwaukee. Those attending came from manufacturers, universities, and computer using laboratories. ... The attendees representing 22 organizations agreed to form a subcommittee of the ACM Computer Languages Committee for the purpose of maintaining and interpreting the ALGOL 60 language. This group is expandable and it is hoped that a European counterpart of this group may be formed so that actions agreed upon by both groups may be regarded as official interpretations and changes to ALGOL 60.

There was a strong feeling among the group that there should not be many changes.

The criteria for new members, by organization, were voted to be the same as were set for the charter members, namely, that members (a) have written, are writing, or plan to write ALGOL-like compilers or are actively engaged in writing programs in the ALGOL 60 language, and of which there are quite a few people writing in

ALGOL 60, and (b) that they are willing to maintain ALGOL 60 as a reference language.

The group then proceeded to get itself a chairman and then took up the proposed changes by Peter Naur, the editor of the ALGOL 60 Report.

The group approved of three of his proposed revisions which are quite minor from the user's point of view but also quite subtle. They rejected one of his proposals and plan to make a substitute for this.

There were also some papers presented at this session: Forsythe on the 'Burroughs Algebraic Compiler and its use for ALGOL programs;' Ingerman on 'Dynamic Own Array Declarations;' Sattley on the 'Allocation of Storage for Arrays in ALGOL 60;' Irons, 'Comments on the Implementation of Recursive Procedures and Blocks;' Ingerman on 'A Way of Compiling Procedure Statements with Some Comments on Procedure Declarations.'

- [56] 60 U.S. ALGOL 60 Maintenance Group Report:
 Sep 12 "Notes on Organizational Rules: The ALGOL Maintenance Subcommittee is in an unusual position because it has a well defined language, ALGOL 60, with which to work. It is important not to do mischief by making major changes, but at the same time interpretations and some changes are necessary. A simple majority vote on a change seems too reckless and a unanimous vote might prevent any action from being taken.
 . . . Eighty percent of the member organizations must repond to constitute a 'proper vote'. If at least 10 percent vote *no*, the proposal is rejected. If the proposal is not rejected and 70 percent vote *yes*, the proposal is accepted. . . ."
- [57] 60 SHARE ALGOL Committee Meeting, M. Bernstein,
 Sep 13-15 Chairman.
 After a call for working volunteers the meeting was declared closed and all others asked to leave. It was agreed to produce an experimental translator based on work that IBM Applied Programming had already done. Mr. Bernstein reported to SHARE that:
 "In line with its original stated purpose, the SHARE ALGOL Committee met last May. Although several positive steps were achieved during the meeting, it appeared that implementation of ALGOL as a SHARE standard programming language was not feasible. As a result, the Chairman requested that members who are unable to contribute time and effort toward ALGOL implementation resign so that a committee of implementors could be formed. A call for volunteers produced enough manpower to attempt a short-range implementation of an experimental ALGOL translator for the 709/90 based on work already done by IB Applied Programming. It is the Chairman's hope that such a processor can be complete within the year and made available to those SHARE members who wish to experiment with ALGOL as a programming language."

- [58] 60 J. H. Wegstein to the Editor of the *ALGOL Bulletin*:
 Sep "As the enclosed notes will explain, a U.S. ALGOL Maintenance
 26 Group has been formed. We hope there will be an European counter-
 part so that changes to ALGOL 60 that are approved by both groups
 may be published as official interpretations and changes to ALGOL
 60. . . . Please advise me of the European status of a mechanism for
 maintaining ALGOL 60."
- [59] 60 F. L. Bauer and K. Samelson to J. H. Wegstein:
 Oct ". . . We are strongly against forming a similar European group in
 20 parallel to an American one since this might either finally lead to
 two different ALGOLs or be the first step to establishing committees
 on a purely national basis with each country having its own repre-
 sentation irrespective of active membership. . . . As a first step in the
 direction proposed we hereby apply for membership in the ALGOL
 maintenance group. . . . We are somewhat concerned over the
 'change' part of the official aims . . . we would like to be sure that
 all members of the group are fully aware of the fact that in Paris
 all committee members were agreed that for some time to come the
 report should not be touched except in the case that ambiguities
 should arise which somehow must be removed. Therefore we feel
 that all definite changes not necessitated by ambiguities although
 they might and even should be discussed very thoroughly, should be
 shelved for a period of two years at least as far as definite action (or
 rather official approval) is concerned. . . .
- In this connection the project of the "Taschenbuch" of algorithms to
 be published by Springer deserves serious considerations. Preparations
 have now reached a state where the editors are forced to freeze
 the language to be used, which will be described in detail in an intro-
 ductory volume. It is obvious that the ALGOL version thus described
 will have to be used throughout the entire Taschenbuch, and at least
 for the near future any changes in ALGOL would simply have to be
 disregarded. If such changes were made, the people for whose
 benefit both ALGOL and the Taschenbuch were intended in the
 first place, namely the large class of engineers and scientists who have
 to do extensive numerical calculations without knowing much about
 computers and logics, will be the ones to be most seriously incon-
 venienced by the confusion arising out of different versions of
 ALGOL. Obviously all this holds for the algorithms reproduced in
 your Comm ACM department as well. . . ."
- [60] 60 H. Rutishauser to the Editor of the *ALGOL Bulletin*:
 Nov "After reading all proposals and counterproposals to remove the
 15 imperfections of the ALGOL-report I am now convinced that in
 order to avoid utter confusion, we have to *maintain* the ALGOL
 word by word as it stands now. In order to avoid ambiguity we
 simply should not use the elements which are not properly de-
 fined. . . ."

- [61] 60 Working Conference on ALGOL, in Moscow:
 Nov "The conference was attended by representatives from the following
 16 organizations:
1. The Steklov Mathematical Institute of the Academy of Sciences of the USSR.
 2. The Mathematical Institute of the Siberian Branch of the Academy of Sciences of the USSR.
 3. The Computing Centre of the Academy of Sciences of the USSR.
 4. The Computing Centre of the Moscow State University.
 5. The Faculty of Mathematical Mechanics of the Moscow State University.
- The recommendations presented . . . represent the common point of view of all participants:
1. The participants of the conference feel that a continuation of the common work on the perfection and sharpening of ALGOL is necessary.
 2. As to the alternatives raised by Dr. Wegstein we prefer the creation of a European group rather than a fusion with the American group. . . .
 3. We are in favor of the voting procedure proposed by the American group.
 4. The organizations taking part in the working conference on ALGOL express their preliminary agreement to enter into the European ALGOL group. . . ."
- [62] 60 ACM Compiler Symposium, in Washington.
 Nov
 17-18
- [63] 60 Advertisement in *Datamation*:
 Nov/ "ALGOL* now at work for Burroughs Computer Users."
 Dec
- [64] 61 Proposal to the ACM ALGOL Maintenance Subcommittee for a Policy on Changes to ALGOL 60:
 Jan
 1
- "1. For the present, changes to ALGOL 60 which would have the effect of invalidating programs acceptable under the syntax and semantics of the 1960 report shall not be approved unless they are necessary to eliminate logical inconsistency or ambiguity. Removal of ambiguities shall be accomplished in such a way that actual changes in the report are minimized.
 2. Changes to ALGOL 60 which will have the effect of invalidating existing programs shall, however, be considered to determine their utility, their implementability, and their effects upon the validity of existing programs. If found acceptable, they may be given tentative approval, to be confirmed when the time comes for an extensive revision of ALGOL.

3. Changes to ALGOL 60 which would *not* have the effect of invalidating programs acceptable under the syntax and semantics of the 1960 report may be approved whenever it can be determined that they meet the following criteria:

- a. They are logically consistent with the present language.
- b. They either extend the scope of algorithms which can be described by ALGOL, or increase the convenience of ALGOL as a programming language, or permit improvements in the object code which would be produced by a compiler.
- c. No superior method of achieving the same end is apparent.

This statement of policy (proposed) is intended to serve as a compromise between two opposing arguments . . . the first . . . that a language in a constant state of flux cannot be expected to gain acceptance. . . . The second position is that a language which cannot describe common computing and data processing procedures is unlikely to gain full acceptance. . . . There will be strong pressure toward development of extended languages which can cope with various tasks of this type, and unless the ALGOL Maintenance Group is sympathetic towards the needs of such workers, there is likelihood of a second Babel. . . ."

- [65] 61 Advertisement in *Datamation*:

Jan

"The Bendix G-20's simplified programming enables your present personnel. . . . Such a programming system is ALCOM—An algebraic problem-solver based on the international mathematical language of ALGOL. Compatible with the ALGO† programming system for the Bendix G-15. . . ."

† Introduced in 60 Oct.

- [66] 61 Open letter to Bob Bemer, from Rene De La Briandais,
Mar in *Communications of the ACM*:

"As far back as Fall of 1958 I recall your mentioning that if ALGOL were not developed as rapidly as possible, FORTRAN would become an industry standard by default. . . . ALGOL has been with us in spirit for some time now, but that's about all. . . . If it is the feeling of IBM that they do not wish to be accused of dominating the industry in the selection of a new 'standard' and therefore they will wait for the ACM or someone else to make this election, then in my opinion it is the wrong attitude for them to take. . . . Let's have some action."

- [67] 61 Reply to the De La Briandais letter:

Mar

". . . Although ALGOL is admittedly a superior language (it should be, for IBM's own FORTRAN and experimental languages made heavy contributions), FORTRAN is the present workhorse and is operative in a large number of installations and understood by thousands of people. It would be unwise to give the user elegance

and take away productivity and efficiency. . . . Rene asks us to give him ALGOL now in place of FORTRAN. Does he wish to do without the input-output facilities and operating system of FORTRAN? . . . When there exists a language fairly safe from arbitrary change and when both the language and the processors offer enough further advantages to customers to offset the costs of re-education, programming modification, and general dislocation—then we will issue a new system with which the user may choose to supplant FORTRAN. . . . Despite the escape clause of the 'reference language', ALGOL will not really be usable until new input-output equipment exists which will handle the character set directly. This area is under experimental investigation, and the production of acceptable new hardware takes considerable time . . . standards are voluntary and have force only when embodied in specific law. . . ."

- [68] 61 Minutes of SHARE XVI—Report by A. L. Harmon:
 Mar " . . . In connection with this, the ALGOL language has a significant
 22 influence on the direction of the FORTRAN growth. In particular, the present 7090 FORTRAN proposal includes several Algolic features. We feel that this is a proper interpretation of the desire of the SHARE body. In order to continue the joint investigation of ALGOL, this past January we delivered our contribution to the ALGOL Committee in the form of an experimental processor. . . ."
- [69] 61 Minutes of SHARE XVI—Introduction to the UNCOL
 Mar Committee Report:
 22-24 "The precise origin of the UNCOL concept is lost in the mists of time. Indeed, it has been reliably reported by Wagner that 'it was well known to Babbage'. . . . Meanwhile, bigger things were on the horizon amidst the soundings of loud trumpets and great waving of arms—an International Algebraic Language.
- While the general pattern of events leading to the 1958 meeting in Geneva is well known, it is not so widely realized that these same events acted as a catalytic agent in the development of UNCOL. The early history of the effort toward design of this International Algebraic Language—or ALGOL, as it is now called—is worth examination in order to gain insight into the driving forces behind UNCOL. . . . Selected items of the relevant correspondence are reproduced in an Appendix to this report.
- Perusal of these letters shows that while ALGOL was in fact designed in response to the desires of the initiators of the effort, some individuals held objections, *ab initio*, on fundamental grounds to the direction taken by the ALGOL group. At three years distance these objections appear to have lost none of their basic soundness, the proliferation of dialects of ALGOL being the best evidence.
- The position of the recalcitrants was (and still is) simply that problem oriented language universality is neither possible nor desirable; that there should be individually tailored POLs for engineers, nuclear

physicists, cost accountants, global strategists or what have you; and that the *real* problem is the drastic reduction of the manpower and elapsed time required to provide a capability of using a given POL with a given machine. Nevertheless, the Pollyannas had their way and ALGOL was born.

It must be emphasized that those who disagree with the proceedings at Geneva *on the above grounds* have no quarrel with ALGOL *per se*. ALGOL is one of several algebraic, formula translation, problem oriented languages and should be judged on its own merit in this company. The objection is entirely against the highly advertised and quite invalid claim of universality in application."

[70] 61 Minutes of SHARE XVI—General Session:

Mar "Mort Bernstein (RS) moved the adoption of the following resolu-
22-24 tion:

Be it resolved that the following letter represents the current opinion of the SHARE membership. The President of SHARE is directed to send it to the President of the ACM and to the editor of the *Communications of the ACM* for publication.

On request of President Cantrell, Bernstein read the letter referred to, which expressed SHARE'S dissatisfaction with ALGOL and rescinded SHARE'S endorsement and support of the language. (Secretary's Note: The complete text of this letter will be found in Appendix F.7.) After the motion was seconded by Frank Engel (WH), President Cantrell called for discussion, which ensued as follows:

GEORGE TAIT (PP): I feel there are many present who have not given ALGOL a fair shake. I suggest that we do not vote on this letter until the August meeting, as its strong wording has some very serious ramifications.

F. J. CORBATÒ (MI): I think the letter has many controversial statements, and while I agree with many of its points, I would not like to see SHARE, as a body endorse it.

DON MOORE (WD): Tait and Corbatò have expressed my feeling perfectly. I feel that this proposal may be the subject of a mail ballot, without necessarily waiting until August to decide it. I move that the motion be laid on the table. (The motion to table was carried, 67-35.)

FRANK WAGNER (NA) asked whether the request of the SHARE body to IBM to implement the ALGOL processor of SHARE machines was still in effect. President Cantrell replied that it apparently was. Wagner then moved that SHARE rescind any request made to IBM to implement any ALGOL processors. There was no discussion, and the motion was carried unanimously."

(Note: The language of the original proposal was strongly intemperate and will not be reproduced here.)

[75] 61 The Rand Symposium, as reported in the 61 Sep issue of
May *Datamation* :

8 "BEMER: . . . No reasonable mechanism for maintenance seems to exist. No one seems to be able to answer the basic question, 'What is ALGOL?'

WAGNER: . . . It is my opinion that ALGOL will never be a widely used language by programmers in large computing installations outside the universities. It has made its run at the leader and failed. I think it can never muster enough strength for a second run, in the terms in which it now exists. I think, however, . . . that it will perpetuate itself as a language for expressing algorithms. It will exert an influence within the universities and 10 years from now, when people whom it has influenced in the universities are in a position of command within industry, we may then see a successor to or derivative of ALGOL in wide day-to-day use. . . ."

WAGNER: . . . Herb's (Bright) comment that the creators of ALGOL were not stubborn enough in trying to keep it truly universal . . . is unfair. When they came up against something that wasn't there, like input-output, or the ability to make tables, or some of the more subtle ambiguities, they had no one to turn to and they had to get their implementation moving along so they had to make a decision. Mr. A made the decision one way, Mr. B made it another; hence we have dialects. . . .

GALLER: As one of the co-authors of one of these dialects, I'd like to explain why we did as we did. We started to write ALGOL 58 for the 704, and we quickly found such things as having to make parentheses do the job of other things. So we found along the way various places where we had to depart from ALGOL 58. We found things like DO and the blank subscript position to be simply unfeasible to put in through a workable translator. Then too, we found several things that we thought were better than the existing ALGOL, and we put them in. . . . When we got all done, what we had simply wasn't ALGOL . . .

WAGNER: At least you had the decency to call it MAD.

BEMER: I want to defend the ease of using ALGOL. You could take a subset of ALGOL and restrict it in such a way that it would be just as easy to use as FORTRAN. It might be a different form, but these are the choices you make. Roy Goldfinger says that you could, if you wish, start from *Alice in Wonderland* and just by making enough changes write a programming language. . . . You could, if you wish, go the other way. Start with FORTRAN, make a few changes here and there and incorporate the best features of ALGOL. It doesn't matter. Maybe we won't get it through the ACM Subcommittee on ALGOL; maybe then the FORTRAN standard will absorb all this. . . ."

CARLSON: . . . One of our engineers decided that people could indeed be trained to use ALGOL and he sat down and wrote an ALGOL primer. Why the people who wrote ALGOL didn't think of writing a primer to explain all this balderdash, I don't know. It didn't take

him very long. . . . We call this the DuPont Publication Language for ALGOL. . . . The fellow who did this work now writes routines in ALGOL, and because he can't put them on a machine with ALGOL, he rewrites them in FORTRAN. He makes the statement over and over that he winds up doing the job in from one-third to two-thirds the total expended time it would have taken him if he did it in FORTRAN in the first place. . . . He is an experienced FORTRAN man to start with. He finds that the ALGOL language takes care of many of the things that he had always complained about the FORTRAN. . . .

BRIGHT: . . . I think people are ignoring the fact that FORTRAN represented a giant step and ALGOL represents a refinement, a generalization, and a maturing. Without the push that FORTRAN got, it could hardly be expected to have such an effect on the industry.

BEMER: FORTRAN wasn't really such a giant step as far as the language was concerned. This had been done by both Rutishauser and Laning and Zierler at MIT many years before. . . . FORTRAN was basically designed as an experiment in object code optimization. . . . It was a laboratory tool for this and I suppose because it was produced by IBM it suddenly got large acceptance.

WAGNER: Remember another thing, though. It was backed up by a very large maintenance group. You could count on the fact that in eight years or so all the errors would have been removed. Maury has a wonderful set of languages in his various NELIAC Processors, but I wouldn't use them even if he rewrote them for the 7090, because I have no assurance that they will be maintained."

- [76] 61 Joint Users Group, Report of Committee on Communi-
May cations:
8 "... It was agreed by all present that it would be useful if a study could be undertaken to summarize the efforts that are presently being made to implement ALGOL 60. . . .
Mr. Ed Manderfield . . . suggested that an effort be made to define a subset of ALGOL 60 suitable for implementation on 'small' computers. . . ."
- [77] 61 ACM Editorial Staff Meeting:
May "Need for ALGOL primer and other material to explain the
10 language. . . ."
(Annotation on my copy, signed by one of the "Paris 13"—"ALGOL is like the Bible, to be interpreted and not understood".)
- [78] 61 Minutes of the ACM Council:
May "... Bob Bemer reported that the implementation of ALGOL
11 processors was going a lot slower than had been hoped when the original ALGOL language was developed. The Council passed a

motion requesting the President to appoint an Ad Hoc Committee to draft a statement clarifying the current position of the ACM with respect to ALGOL. This draft will be circulated to the Council for approval and if approved will be published."

- [79] 61 The President of ACM appointed Wagner, Forsythe,
May Wegstein and Bemer to an Ad Hoc Committee "to make a
15 recommendation to the ACM Council relative to the situa-
tion on ALGOL."
- [80] 61 First Meeting of ISO/TC97/WG E on Programming Lan-
May guages, in Geneva.
18 Following plenary sessions of the International Standardization
Organization's Technical Committee 97 on Computers and Infor-
mation Processing (also the first meeting), the newly authorized
Working Group E on Programming Languages met under the
chairmanship of R. F. Clippinger, as the U.S. holds the Secretariat.
Following national activity reports, the Working Group decided to
take the first actions on that portion of its scope which read:
"Collect documentation for, classify and catalog existing langu-
ages and their applications."
- [81] 61 P. Z. Ingerman to the ACM ALGOL Maintenance
May Group, Proposed Alternative to the Oak Ridge Proposal:
31 "The members of this group will adhere to the ALGOL language as
defined in the ALGOL-60 report. Translators should be constructed
in such a way that ALGOL programs that are unambiguously de-
fined by the report will be clearly translated. The committee will
prepare immediately a list of ambiguities at present in the ALGOL
language so that these ambiguities may be avoided by algorithm
writers who prefer quiet to contention. . . ."
- [82] 61 F. V. Wagner to "Those Concerned With Implementing
Jul ALGOL For Computer Manufacturers":
20 ". . . The National Council of ACM believes that it is important for
it to review and clearly define its present policy in connection with
ALGOL. I have been appointed chairman of a committee whose
function is to draw up a proposed statement of policy for the con-
sideration of the National Council. It is important that this com-
mittee be aware of the present plans of all computer manufacturers
for providing ALGOL processors for their various machines.
The Committee would appreciate it very much, therefore, if you
would assist them in their task by sending to each member of the
committee, listed on Enclosure (1), the following:
- (a) A formal statement as to your company's plans for providing
ALGOL processors. . . .
 - (b) Any written material which defines as thoroughly as possible
the *exact* form of input language that would be acceptable to
those processors, and its meaning to those processors.

- (c) Your opinions as to the strong points and deficiencies of ALGOL, both from the point of view of the *user* of the language as well as the system programmer who is designing processors to accept it. In addition, if you have any opinions as to the policy that the ACM should follow, or action that it should take, we would be interested in knowing about them. . . ."
- [83] 61 H. R. J. Grosch, in *Datamation*:
Jul "... But the various ALGOL groups ought to agree on just one thing, just once, and head for the Elephant's Burial Ground. . . ."
- [84] 61 RCA to Members of the ACM ALGOL Maintenance
Jul Group:
19 "We support the sentiment expressed in the Oak Ridge proposal . . . we request the chairman to call for a vote on the above mentioned proposal."
- Membership:
- | | |
|--------------------------------|--------------------------------|
| Armour Research Foundation | Princeton University |
| Bendix Computer Division | Remington Rand Univac |
| Burroughs—Electrodata | System Development Corporation |
| University of Calif., Berkeley | Stanford University |
| Case Institute | Sylvania Electric |
| University of Chicago | Computer Associates |
| Georgia Tech | RCA |
| Lockheed Aircraft | Carnegie Tech |
| National Bureau of Standards | University of Mainz, Germany |
| U.S. Naval Electronics Lab. | Argonne National Laboratory |
| University of North Carolina | Royal McBee Corp. |
| Northwestern University | DuPont |
| Oak Ridge National Laboratory | IBM |
| University of Pennsylvania | Dartmouth College |
- [85] 61 Jean Sammet of Sylvania voting NO on Oak Ridge Pro-
Jul posal:
28 "... problems do not disappear just because they are ignored. I consider the most objectionable sentence in the Oak Ridge proposal to be the one stating: 'For several years to come this committee will not propose any changes or additions to the ALGOL language.' This seems to negate the very purpose of having a Maintenance Committee . . . either a problem exists or it doesn't, and in the former case it should be solved. . . . It must be emphasized that there is a difference between doing things slowly and carefully and not doing them at all. The ALGOL Maintenance Committee seems to be heading in the latter direction, whereas it could so easily be taking the steps which are necessary to improve the usage and acceptance of ALGOL as a universal language."

- [86] 61 Computer Associates voting NO on the Oak Ridge Proposal:
Aug
2 "... We agree with Ingerman that 'anything which is ambiguous is undefined' is an unsuitable answer to the ambiguities of ALGOL 60. . . ."
- [87] 61 H. Rutishauser, in the *Automatic Programming Information Bulletin*:
Aug
"... I must recall that ALGOL is not just a programming language, but an internationally accepted standard notation, for which any change has rather severe consequences. . . ."
- [88] 61 J. H. Wegstein's "ALGOL 60—A Status Report", published later in the 61 Sep issue of *Datamation*:
Aug
7 "... This report was written in response to recent intimations that ALGOL is or should be on the wane. One is reminded of Mark Twain's response to rumors that he had died. 'The reports of my death have been greatly exaggerated.'
Physicists define momentum as equal to mass times velocity, and it is impossible to estimate the momentum of an object by observing only its velocity. A very massive object may have a large momentum even though it is moving very slowly. Similarly with ALGOL, the momentum of the movement cannot be judged by the speed with which the language is being put into use without also observing the number of people who are working with it.
At this time, the future widespread use of ALGOL for publication and teaching purposes seems certain. It is rather easy to translate by hand from ALGOL into various computer languages or into other artificial languages similar to ALGOL for which compilers now exist. The permissibility of many hardware languages that are only similar to the ALGOL publication language may be essential to giving the publication language a foothold. Yet, as time goes on, the urge to 'stand closer to the trough' will surely lead to compilers which bring the computer very close to the ALGOL publication language."
- [89] 61 IBM Reply to the Wagner Questionnaire:
Aug
14 "A. IBM expects to supply, at some future time, processors that accept languages of the ALGOL class for such of its machines that it may be practical. We do not wish to make premature disclosures, but we may say that we are pursuing several compatible approaches—including the following, about which you are familiar:
1. Improvements and modifications in the FORTRAN language to incorporate the new and desirable features of ALGOL. These are reflected in the specifications for a 7090 processor.
2. Experimental investigation and study of the properties of such languages and their translators."

3. Cooperative participation in the SHARE/ALGOL committee. Our main contribution so far has been the experimental processor for the 709/7090 of about 18,000 instructions.
 4. Participation in the ALGOL 60 maintenance group chaired by Mr. Joseph Wegstein.
- B. No written materials are available other than the documentation furnished to the SHARE/ALGOL Committee.
- C. We feel that the strong points of ALGOL are self-evident and that the deficiencies have been adequately noted in:
1. The *ALGOL Bulletin*, Copenhagen.
 2. The *Communications of the ACM*.
 3. The notes of the ALGOL 60 Maintenance Committee."
- [90] 61 Letter from R. W. Bemer to I. L. Auerbach, President of
 Aug IFIPS:
 15
- "... IBM's feeling that the maintenance of ALGOL should be undertaken at an international level reflects the curious impasse facing the ALGOL 60 maintenance committee. The rules of this committee are such that a negative vote of 10% or more is sufficient to defeat a resolution. Accordingly, the present resolution is defeated. However, if a proposal to make a specific change in ALGOL were submitted to the committee, the members now voting against changes in ALGOL (a majority, although not enough to pass the resolution) would constitute a body of more than 10% required to defeat a proposal of this type. Thus the committee finds itself, as a result of this vote, in a curious position. IT CAN'T CHANGE ALGOL and IT CAN'T NOT CHANGE ALGOL. This conclusion was confirmed by Mr. J. Wegstein in a phone conversation with me.
- Since this committee, by all laws of logic, can produce only zero output, it would seem that an appropriate international committee with authority is necessary to maintain the language. In Mr. Utman's letter to you, on behalf of the secretariat of Working Group E of TC97, he stated that both Tootill and I did not mention specific IFIPS interest in programming languages at the Geneva meeting. I think this position was correct as we were not instructed to do so, and indeed your reply to Utman supports this. However, the original sponsors of ALGOL are now components (in one form or another) of IFIPS and you might find it necessary at some future time to re-evaluate the IFIPS position. . . ."
- [91] 61 IBM voting NO on the Oak Ridge Proposal:
 Aug "We feel that the ALGOL language should be maintained. However,
 16 we would wish such maintenance to be carried on by a unified international committee sponsored by an authoritative international body such as IFIPS or ISO.
- Our vote is based on the fact that no such international body exists with the authority to maintain ALGOL. . . ."

- [92] 61 Minutes of SHARE XVII—Motion to Withdraw Support
Aug ALGOL:
23

"The question of ALGOL 60 was re-introduced by Mort Bernstein (RS), who moved to call from the table and amend slightly his ALGOL resolution made at SHARE XVI. The effect of this resolution would be to withdraw the support of SHARE, as a body, from the ALGOL effort and to notify the ACM of this. . . .

Bernie Rudin (ML) pointed out that as a result of recent work, ALGOL was more nearly complete than the resolution would indicate; nevertheless, he said, the ALGOL Committee would have no real objections to the proposed letter. Frank Wagner (NA) stated that, as chairman of an ACM Committee to study policy on ALGOL, he was no longer permitted a personal opinion; however, he suggested that a paragraph be added which would take account of the recent developments in ALGOL. J. A. Buckland (SO) felt that the letter was incomplete and inaccurate, and F. J. Corbato (MI) objected that it was gratuitous and could place SHARE in a false light in the eyes of non-members. Aaron Finerman (RF) reminded the body that it had endorsed ALGOL three years earlier and that the intent of the letter was to inform the ACM that SHARE no longer approves ALGOL wholeheartedly.

The motion was put to a vote and carried, with 65 installations in favor, 43 against, and 15 abstaining."

- [93] 61 Letter from the SHARE President to the President of the
Aug ACM:

"In September 1958, at the 11th meeting of SHARE, a resolution commending the efforts of the ACM-GAMM IAL Committee was unanimously approved and SHARE adopted ALGOL as a language for SHARE machines. SHARE prevailed upon the vendor of its machines to produce an ALGOL processor under the direction of the SHARE ALGOL Committee. In the next year the SHARE ALGOL Committee proposed a number of extensions to ALGOL, and recommended to ACM that a mechanism be established for the recognition of the continued development and extension of ALGOL for the purpose of establishing standardization among all computer users.

By 1960 enthusiasm for ALGOL within SHARE had begun to wane, and the work of the ALGOL Committee was frustrated by apathy. The Committee was reorganized at the 15th meeting of SHARE, with only those members pledged to work on ALGOL implementation remaining. The goal was set to produce an ALGOL processor for the 709/7090 by September 1961. In six months, this effort also failed to make any significant progress.

With this background, at the 16th meeting of SHARE, a resolution was passed which withdrew SHARE'S previous request that IBM produce an ALGOL translator for SHARE machines. Among the reasons for this action were the following:

1. It has been impossible to generate sufficient enthusiasm for ALGOL within SHARE to ensure its implementation on SHARE machines.
2. The SHARE ALGOL Committee has reported that ALGOL 60 seems to be incomplete, ambiguous, and difficult to implement in its entirety, and that there does not exist an effective way of resolving the troublesome issues.
3. The ALGOL dialects which have resulted from various attempts at implementation on several different non-SHARE machines, while being ALGOL-like, still do not retain the compatibility of source language which it was hoped ALGOL would achieve.
4. FORTRAN has become a generally accepted and well known algebraic system for which processors exist on SHARE machines, as well as many other computers.

While hereby acknowledging the inability of SHARE to obtain a working ALGOL 60 processor as a successor to FORTRAN, this is done without prejudice to the efforts of those members of SHARE who wish to continue to experiment with, develop and implement ALGOL 60."

- [94] 61 J. H. Wegstein, Chairman, to the Members of the ALGOL
Aug 60 Maintenance Group: Vote on the Oak Ridge Resolu-
28 tion:

"... Since more than 10% voted *no*, and less than 70% voted *yes*, the motion does not carry. However, one might observe that by the same rules, as long as those who voted *yes* do not change their minds, no changes to ALGOL are likely to be accepted."

Actual vote—16 for no change, 10 for OK to change, 2 missing.

- [95] 61 ALGOL Maintenance Group Meeting, in Los Angeles.
Sep
5

- [96] 61 Minutes of the ACM Council:
Sep
8

"... Frank Wagner presented the report of the Ad Hoc Committee on ALGOL. After reviewing the history of ACM's participation in ALGOL, he reported that the committee had sent letters to the larger manufacturers and users' organizations. The replies to these letters showed that the manufacturers varied between those who were extremely enthusiastic to those who were taking no announced action at the present time. After considerable discussion, the following resolution was passed:

'BE IT RESOLVED that the Council of the ACM adopt the following policy with regard to ALGOL and direct that it be published in the Communications of the ACM:

1. The ACM supports ALGOL 60 as the preferred publication language for appropriate algorithms.
2. The ACM continues to encourage research into development and evaluation of languages for publication and programming.

3. The ACM believes that ALGOL 60 is a language worthy of consideration by national and international standardizing bodies.'"

[97] 61 Working Conference on Automatic Programming Methods, in Warsaw.
Sep

5-13

Attended by about 60 representatives from the Soviet Union, Czechoslovakia, German Democratic Republic, Hungary and Rumania.

[98] 61 News item in *Datamation*:

Oct

"IBM's ALGOL is not available: Posted on the bulletin board at the recent ACM conference in Los Angeles was a listing of all compilers presently completed, their completion dates, and the machines for which they were prepared. The tabulation was presented and posted by IBM's Bob Bemer and included the attention-getting fact that an ALGOL processor for the 7090 was completed by IBM as of December, 1960.

Although this was assumed by many registrants as an announcement of availability (although indeed, a curious one), this is not the case. The processor which was prepared and listed as completed was written on an internal, experimental basis for educational research only. It is likely that if an ALGOL processor was developed for IBM users, it would not be this one.

However, field testing of this ALGOL processor will take place by a number of SHARE ALGOL committee members early next year. And while testing is hardly to be considered an IBM endorsement of ALGOL or the outdating of FORTRAN, progress in this direction is interesting to note in view of the following excerpt:

In the March 1961 issue of the *Computer Bulletin*, publication of the British Computer Society, an article on 'Survey of Modern Programming Techniques,' by R. W. Bemer was published and we quote in context from p. 130: 'I have enough faith in the eventual future of ALGOL to have caused a program to be constructed which converts from FORTRAN source language into a rather stupid ALGOL. I have been asked many times why we did not make it translate from ALGOL to FORTRAN so that the existing processors could be utilized. The answer has always been that we wish to obsolete FORTRAN and scrap it, not perpetuate it. Its purpose has been served. . . .'

(One could of course note that this talk to the BCS was given in 60 September, at which time SHARE had not convinced IBM to change its ALGOL policy—or that the survey was done for the ISO/TC97, . . . or that we were speaking of poor processors which should be improved—in any case, it's an easy start on the road to the Research Division.)

- [99] 61 C. J. Shaw, in *Datamation*:
 Oct "JOVIAL is a procedure-oriented programming language derived from ALGOL 58 and designed by the System Development Corporation for programming large computer-based command/control systems. JOVIAL is largely computer-independent; compilers for the IBM 709/7090, the CDC 1604, the Philco 2000, the AN/FSQ-7 and the AN/FSQ-31 are currently in operation or in check-out. . . . This flexibility is due to the fact that JOVIAL compilers are written in JOVIAL, in a computer-dependent and, to a lesser extent, system-independent form. . . ."
 (JOVIAL is an acronym for Jules (Schwartz) Own Version of the International Algebraic Language. A very complete historical paper is the entire content of *APIB* 22, 64 Aug, again by C. J. Shaw.)
- [100] 61 Meeting of the IFIP Council, outside of Copenhagen:
 Oct An unanimous vote authorized a Programming Languages Committee, TC 2. The Council was to suggest candidates for the Chair. (Dr. H. Zemanek of IBM Vienna was named.)
 23-25 (From my notes: Bauer said that ALGOL was a product of a combined effort of representatives of technical societies and the language has status for this reason only. van Wijngaarden said that such a language has status only by general acceptance. I then submitted the hypothetical case of someone publishing, under the auspices of one or more technical societies, a revision of ALGOL as a new language specification. I asked if this would be proper and would such a language replace ALGOL if it got equivalent or greater acceptance (I just happened to have the specs for 'IBM ALGOL' with me)? There seemed to be a general feeling that this was not quite a cricket thing to do. . . . Bauer and van Wijngaarden were in agreement in their insistence that only the thirteen original authors could re-issue or legally modify ALGOL. . . . Bauer, on the basis that my English was better than his, asked me to write a draft letter to the original authors of ALGOL. . . . this was tested, in the writing, with van der Poel. . . . An informal group met on Tuesday. . . . to consider the next steps. By this time there was considerably more understanding of IBM's position and what could be done. . . . It was at this time that van Wijngaarden had a flash of perception about software penalties and lack of rental being a major problem to any computer manufacturer. There seemed to be a general feeling that the clean-up effort should be made in order that IBM could become an active aid in the ALGOL movement. . . .")
- [101] 61 R. E. Utman, in *Datamation*:
 Nov "... specifications have been found ambiguous or impractical of achievement. In this condition they permit such varied interpretations that some of the resultant processors can hardly qualify to carry the names ALGOL or COBOL. Yet they are being labelled and sold as such. . . . This need in information processing was recognized in 1960 by ASA, and the responsibility for programming languages established within the scope of Sectional Committee X3 and its

Subcommittee X3.4. Under the Chairmanship of Dr. J. Chuan Chu, a year of education and experience has since accrued and significant progress can be reported today.

The first thing the programming experts in X3.4 believe they have learned is that the problem of standardizing a language seems several orders of magnitude more complex than that of a unit of measure, a railroad gauge, film size, etc. . . . As in every field of technology, a standard must be dynamic and maintained in order to be realistic, useful, and accepted. . . . Once the standard language is achieved, it will then be necessary to specify tests to be used in qualifying the variety of interpretations that will be labelled and sold in its name. There must be an organized discipline of some sort to enforce evaluations by these tests, and to administer a continuing program of certification. . . ."

[102] 61 F. L. Bauer and K. Samelson to the Authors of the
Nov ALGOL 60 Report:
30

"Nearly two years have elapsed since the issuance of the ALGOL 60 Report. Processors (translators) have been written for many machines during this period, resulting in many advances in translation methods and considerable experience in the actual writing of such processors. In addition, there now exists some experience in using ALGOL for the machine solution of problems and even more experience in the communication of algorithms in the publication language. Thus the overall acceptance of ALGOL, as a language for scientific problems, has been good, and especially favorable in Europe.

However, some people claim that there are some obstacles to the general acceptance of ALGOL. Indeed the ALGOL Bulletin and various working groups have served as a forum for discussion and suggestions of interpretation, revision and extension. None of these methods have proved sufficiently effective against minor variance in both language usage and processor interpretation, possibly caused by the report and not due to deliberate intention.

It has been suggested that some minimum amount of resolution is necessary and that the most effective (and at present the most authoritative) means of doing this is the reconvening of the original committee, as discussed at the end of the Paris meeting. Therefore we request you to consider your participation in such a meeting and to secure the acceptance of your attendance from your sponsor.

It has been suggested that an original member may not be deeply concerned with ALGOL now and therefore may not wish to participate. Although participation is not mandatory, it would be helpful to have a letter of resignation so that the authority of this body is not diminished.

Furthermore, it might be desirable to invite a few additional members who are acknowledged, practicing experts. It has been indicated that it may be highly desirable that a conference member may be accompanied by a technical advisor to him. In this way, some

- processor implementors can be brought into useful contact with the conference.
The meeting should occupy three days, starting Monday 2nd April 1962, immediately after the ICC Symposium on Symbolic Languages in Rome . . ."
- [103] 61 Date of internal report "IBM ALGOL—a revision and
Dec modification of the ALGOL 60 Report due to an ad hoc
5 IBM committee. . . ."
- [104] 61 P. Naur to the authors of the ALGOL 60 Report:
Dec "In reply . . . I can say that I agree that the time is ripe for a removal
8 of ambiguities and omissions in the ALGOL 60 Report. However, I regret that I cannot at present support the suggestion of settling these questions through a meeting of, essentially, the original committee. . . .
The formation of the U.S. Maintenance Committee and the result of the enquiry of the ALGOL Bulletin, particularly the unchallenged conclusions AB 11.1.6, make this approach impossible as far as I am concerned, at least at the present.
I would like to add that I have already for some time been working on a different approach to this problem . . . the first step . . . is the distribution of a detailed questionnaire in the ALGOL Bulletin. . . ."
- [105] 61 K. Samelson and F. L. Bauer to the authors of the
Dec ALGOL 60 Report:
13 "... We therefore consider Peter Naur's material as a very useful contribution to the list of suggested topics, but stand to our request from Nov. 30 of a meeting of the original committee."
- [106] 61 News item in *Datamation*:
Dec "Frankly acknowledged by many IBMers as a far superior processor to FORTRAN, ALGOL development is nevertheless far from practical in the eyes of IBM management. The problem is not one of money but largely the lack of experienced programmers to meet present commitments for over 35 FORTRAN processors as well as numerous other dialects promised to IBM customers. In addition, scrapping their present investment in FORTRAN would involve an enormous risk for IBM with no national or international body providing the needed authority for a definitive explanation of ALGOL. . . . The current status at IBM: considerable head-scratching."
- [107] 61 D. D. McCracken, in *Datamation*:
Dec "... I think it's time somebody spoke up for the power of ALGOL in doing 'ordinary' programming—the kind of work in which recursive definition, 'own' variables, and call-by-name seldom arise. . . ."

It is interesting to speculate on the origin of the myth of ALGOL's abstruseness, for which I suggest three reasons. First, the report . . . is excellent for its intended purpose of defining the language, but somewhat lacking when viewed as a beginner's primer. . . . Second, most of the published discussion of ALGOL has centered around the advanced features, which is entirely reasonable, but misleading . . . this leaves those of us on the fringes with the entirely mistaken impression that ALGOL consists only of the difficult things. Third, the algorithms published in the *Communications* are slow going for some of us *because the problems they solve* are slow going for some of us . . . in the process of exhibiting how ALGOL can be used for difficult problems, some of us got the impression that that was the whole story.

In summary, it appears to me that ALGOL offers clear-cut advantages to anyone doing scientific computing, whether or not the application requires use of the more advanced features of the language. These features may well turn out to be major advances in the computing art; in the meantime, there is no need to wait for the dust to settle before making use of the 'simple' advantages. . . . It's time for some of us to take a fresh look."

- [108] 61 Minutes of the ACM Council:
Dec
14 "Resolved: That the ACM request AFIPS to request IFIPS to reconstitute the ALGOL Maintenance Committee under the auspices of IFIPS."
- [109] 62 K. Samelson to P. Naur and members of the ALGOL 60
Jan
5 Committee:
". . . For the real crux of our problem is to convince the leading computer manufacturer(s) to incorporate ALGOL processors in the programming systems they provide for their products. This requires a clearcut unambiguous language presented with clear authority. If this is not available some manufacturers will continue to give their own interpretation to ALGOL. Others will continue to be disinclined to incorporate ALGOL in their programming systems, and the story of SHARE ALGOL indicates that no exercise in group dynamics will change that. . . ."
- [110] 62 Issuance of *ALGOL Bulletin* No. 14, containing the Naur
Jan
12 Questionnaire.
- [111] 62 D. D. McCracken, in *Datamation*:
Jan
". . . Despite its demonstrable advantages as a computer language, ALGOL will gain acceptance slowly (but steadily). Acceptance would be much more rapid if users were willing to believe that (a) ALGOL has not already been engraved in granite, never to be changed, and (b) it will not change so drastically every two years that processors will be continually obsolete. It would also help,

of course, to hear a little more enthusiasm from the direction of White Plains. Maybe we will have to wait for FORTRAN to evolve into ALGOL, as it already appears to be doing. . . ."

- [112] 62 F. V. Wagner, in *Datamation*:
Jan " . . . For general purpose work, FORTRAN will continue to maintain its supremacy. It will have little competition, except in universities, from any of the ALGOL variants. . . . Thus we can look for an increased pressure for the incorporation into FORTRAN of features permitting the easy development of special-purpose POL's within the FORTRAN system. If FORTRAN does not rise to meet this challenge, it is possible that the pendulum may swing to one of the dialects of ALGOL. JOVIAL is the most likely candidate. . . ."
- [113] 62 A. J. Perlis, Chmn., ACM Programming Languages Committee, to the Authors of the ALGOL 60 Report (American Delegation):
Feb
15 " . . . The revision of the report is quite important for political, as distinct from practical, reasons. It is important that the American delegation do their part in aiding the adoption of ALGOL as a computer language by at least removing any impediments due to ambiguities within the ALGOL 60 report.
Thus, will you please inform me at the earliest possible time of your intent to attend the meeting . . . or send me a letter of resignation from the Report Committee. . . ."
- [114] 62 J. H. Wegstein to the Authors of the ALGOL 60 Report:
Feb
19 " . . . Although a declared international standard may be years away, nevertheless questions are being asked of ALGOL and its ambiguities. From a practical point of view, I think that ALGOL 60 could be used as it is for two or three more years. From a political point of view there needs to be a 'flawless' ALGOL and an organized ALGOL supporting group. . . . Let us see if errors can be corrected, results can be accepted, and set the course for establishing a permanent ALGOL maintenance group."
- [115] 62 B. A. Galler to the Editor of *Datamation*:
Feb
"One of my colleagues has pointed out to me that the best argument one could give for switching from FORTRAN to MAD, ALGOL or BALGOL is the cover of the December issue. The simple iteration pictured in all four languages is correct in MAD, ALGOL and BALGOL, but in FORTRAN we find not only a mixed expression, but incorrect formation of the 'DO' statement (for the iteration desired)."
- [116] 62 Prof. T. A. Gallie, Jr., to W. C. Hume, Pres., IBM Data Processing Div.:
Feb
21 " . . . IBM took an active part in the birth of ALGOL. . . . Applied Programming is having trouble making FORTRAN work on some

computers (such as the 7070) and hence is hesitant to tackle ALGOL. Duke University has written and is happily using an ALGOL compiler for the IBM 7070. We much prefer this language to FORTRAN and our translator is many times faster than Applied Programming's basic FORTRAN compiler which, in turn, is many times faster than their full FORTRAN compiler. On the other hand, our object programs are probably half as fast as those produced by FORTRAN and therefore of no use to many FORTRAN users. However, we would like to make ALGOL available to the many 7070 customers . . . who have told us they want it.

The problem is that IBM can't decide what card punches should be assigned to a few additional characters. . . . More precisely, we want one 026 printing keypunch with a few extra characters which *have IBM's blessing*, so that other IBM customers will eventually rent similar keypunches and possibly use our ALGOL translator. . . ."

[117] 62 W. C. Hume to Professor T. A. Gallie, Jr.:

Mar "I should like to take this opportunity to congratulate you on your
9 ALGOL compiler for the 7070 and to assure you that we consider it to be in IBM's interest for the ALGOL language to be implemented as quickly as possible. . . ."

[118] 62 First Meeting of IFIP TC 2, Programming Languages, in
Mar Feldafing (Munich):

20-21 "1. The scope of the committee shall be to promote the development, specification, and refinement of common programming languages with provision for revision, expansion and improvement.
2. The specific program of work shall include:
(a) General questions on formal languages, such as concepts, description and classification.
(b) Study of specific programming languages.
(c) Study and if appropriate coordinate the coalescing of a new programming language for which there appears to be a need.
3. The Technical Committee may request the establishment of working groups if and when appropriate.
4. The Technical Committee shall establish and maintain liaison with other appropriate international organizations.

.....

1. A working group may be established by the Council of IFIP upon the request of a Technical Committee. It is a group of technical experts selected without consideration of nationality and assigned to work in a specified technical area.
2. The membership of a working group is appointed by the chairman of the corresponding Technical Committee with the approval of this Technical Committee. Membership is not restricted to persons who belong to IFIP member societies or groups of societies.
3. The chairman of the working group is appointed by the President of the Council with recommendation from the Technical Committee.

4. Publication of results in the name of the working group can be made after having been reviewed by the Technical Committee under the provision that explicit mention will be made of the fact that it will be submitted for approval at the next Council meeting. After this approval it becomes an official IFIP publication."

Working Group 2.1 on ALGOL was established under the chairmanship of W. L. van der Poel:

"The working group will assume the responsibility for the development, specification and refinement of ALGOL."

D. D. McCracken, writing in the 62 May issue of *Datamation*:

"ALGOL has a home. . . . This is indeed important news to anyone interested in the acceptance of ALGOL, since one of the main obstacles to its adoption has been its homelessness. Until now, no one could really speak for ALGOL with complete authority except the 13 authors of the original report and they were not in the language-maintenance business. Now there will be an official body to which questions, suggestions, and complaints can be directed, with assurance that a response will be forthcoming and that it will be official policy. . . ."

- [119] 62 Symposium on Symbolic Languages in Data Processing,
Mar in Rome, sponsored by the International Computation
26-31 Centre (UNESCO) with published proceedings:

D. D. McCracken, reporting in 62 May issue of *Datamation*:

". . . Condensed to essentials, the argument ran: 'We've got a lot of customers who need *answers*, not speculation. We would be happy to use ALGOL, since it seems to have many good features, but we can't do much with a compiler that is loaded down with these miserable recursive procedures and which produces horribly inefficient object programs. We want to *work* with ALGOL, not *play* with it.' There was loud, sustained applause.

Four viewpoints could be identified in the ensuing discussion. Some one said: 'But I've got a compiler that isn't slowed down by recursiveness, and the object programs are pretty good. You've just got to learn how to write compilers.' Somebody else said: 'Maybe recursiveness does cost time in some cases, but it costs *not* to use it when it is the best solution. You've just got to learn how to use this new tool we've provided.' Another said: 'Even if recursiveness is difficult and often not useful, the idea of ALGOL for standardization is so important that some compilers should be constructed without recursiveness, if necessary. You've got to provide us with more than one version of ALGOL.' Finally, someone said: 'ALGOL is such a large advance in the computing art that we never should

have expected immediate acceptance. We've got a lot of things to learn before ALGOL is widely accepted, as it surely will be in time, and one of these is patience.' . . ."

(The proceedings of this symposium contain verbatim records of the panel discussions, all of which will be very interesting to the new worker in this field. I doubt if there will be many advances by 1975 for which the germ of the idea cannot be found herein. This quandary must be resolved by referencing, not actually duplicating in this log, although many of the comments noted are more important intrinsically than many of the elements of this log.)

- [120] 62 Continuation of the first meeting of IFIP TC 2. ALGOL
Mar Working Group 2.1 authorized, in Rome.
27

- [121] 62 Meeting of the ALGOL 60 Report Committee, in Rome,
Apr resulting in the Revised Report.
2-3

Approximately 30 minor changes were agreed. Basic input were the results of the questionnaire in *ALGOL Bulletin* 14 and proposals and ambiguities described in that publication and the Communications of the ACM. Incorporated in the report was their acceptance of transfer of responsibility for the language to IFIP WG 2.1.

As it turned out, 8 of the original authors participated, 1 direct representative of an original, 6 advisers and 1 observer; the last being van der Poel, who had to take over the responsibility for IFIP.

Two major disagreements were noted. Naur, van Wijngaarden and van der Poel were in favor of no distinction between procedures and functions, together with the concept of body replacement in the procedure definition. Opposed were Bauer, Samelson, Green and Kogon.

- [122] 62 Conference on Advanced Programming Languages for
Apr Business and Science, at Northampton College, London.
17-18 Proceedings published in the *Computer Journal*. Some excerpts from the discussions:

"A. GEARY: . . . My first pleasure is to introduce Dr. Dijkstra. . . . We have warned him that there has been a certain amount of bias against ALGOL in England, in some quarters. . . .

G. M. DAVIS: . . . We might also agree on the standardization of means of specifying and describing languages. Until this is done one cannot start the standardization of languages themselves. . . .

M. V. WILKES: . . . It will in the future be useful to know exactly what is meant by a given programming language. . . . But to suggest that standardization should mean the selection of one particular language to be used on all occasions, in preference to all others, appears to me to betray a very superficial knowledge of the subject. . . .

E. W. DIJKSTRA: . . . The main virtue of recursive procedures is that they make the tool more lovable for computers. A few weeks ago somebody used the phrase 'ALGOL playboys' in a nasty fashion, and I was very angry. A Dutch philosopher wrote a big book called *Homo Lucidas*†—the plain man—showing clearly that everything which, ages later, was regarded as of some significance, had started off as being 100% plain. . . ."

†(Note: M. Halpern suggests that the transcript was not verified by the participants, and that the book was in fact *Homo Ludens*—The Playing Man. The last word would thus be "playing".)

- [123] 62 Invitations for Membership in IFIP Working Group 2.1,
Apr ALGOL, tendered by Prof. H. Zemanek, Chairman of
24 IFIP Technical Committee 2, Programming Languages.
- [124] 62 G. E. Forsythe to the ACM Editorial Staff:
May "... It was agreed that the ALGOL movement has progressed to
7 the point where it is no longer desirable to publish unrefereed
algorithms. Perlis stated that beginning with 1963 the algorithms will
be refereed . . . Perlis formulated the following policy:
a. The Communications will publish codes in any language as
part of a refereed article.
b. In the Algorithms section the codes must be in ALGOL (or
COBOL?)"
(This answered some pressure from SHARE to publish algorithms
in FORTRAN.)
- [125] 62 ISO/TC97/WG E meeting, in Stockholm, courtesy of the
May Swedish Standards Commission, Olle Sturen, Director.
9-10 Survey of Programming Language Processors presented as (USA-
10)55, 7 pp. Later published in *CACM*, 63 Mar.
- [126] 62 One week Symposium at the London School of Economics.
Jul Proceedings published as—Wegner, P. (Ed.) *An Introduction to
Systems Programming*, Academic Press, 316 pp.
- [127] 62 Meeting of U.S. Participants in IFIP WG 1, in New York
Jul City.
10 It was agreed that there was no need for independent formulation of
a U.S. position.
- [128] 62 *Datamation* interviews W. C. Hume and A. L. Harmon
Jul of IBM:
"Q: . . . the American Standards Association, the Association for
Computing Machinery, and the International Federation of Infor-
mation Processing Societies, as well as several other manufacturers,
have advocated the implementation of ALGOL. What is IBM's
position?"

HUME: May I ask a question? What's the stand of GUIDE and SHARE who are the major machine users?

Q: In support of FORTRAN.

HUME: And logically, because they have a tremendous investment. I think one of the successes of IBM has been based on the fact that we try and service the investment of our customers. That's number one. These are the major users of the machines. Secondly, I somehow feel that there is a wrong impression as to our support of ALGOL. We are not ignoring ALGOL. We're really taking a look at it and, over and above a look, we're putting a tremendous investment in ALGOL. Some people have the feeling that just because we're continuing FORTRAN with the investment that our customers have in it and since GUIDE and SHARE have come out for FORTRAN, that we're against ALGOL. We're not against it. We're simply saying that we have to support FORTRAN.

Q: By investing in ALGOL, do you mean research into the development of an ALGOL processor?

HUME: Yes.

Q: Will it be announced soon?

HARMON: As you know, ALGOL and its specifications are still under development and we have submitted an experimental ALGOL processor to the SHARE ALGOL committee for further development and work. We like to make sure that things are reasonably cleaned up before significant assets are poured into any program.

Q: Regardless of an announcement date of an ALGOL processor, would this signify the end of FORTRAN maintenance and development?

HUME: It would not.

Q: With this fact in mind, would you be able to provide a prediction as to when an ALGOL processor might be forthcoming from IBM?

HARMON: You're in an area where you're almost asking when something will be invented. In a development program, it's extremely difficult to forecast even close to when something will be specified to the level where it can be properly implemented. My guess would be that the next five years will show significant changes, not only in the ALGOL effort, but also in the FORTRAN language itself. It's conceivable that these two will marry. . . ."

[129] 62 News item in *Datamation*:

Aug

"The winner in a hotly contested language wrestling match is . . . JOVIAL, at least as far as the U.S. Navy is concerned. . . .

The JOVIAL adoption is opposed by NELIAC advocates who contend that their language was originally designed for the Navy and could be used more easily by personnel of less experience than would be required for JOVIAL. In addition, a recent study . . . indicated much faster compiling and executing speeds for NELIAC over JOVIAL."

(Comment: ALGOL variant A vs. ALGOL variant B.)

- [130] 62 Meeting of IFIP TC 2 (second meeting), in Munich.
Aug
25
- [131] 62 IFIP Congress 62, in Munich. Reported in 62 Oct *Datamation*:
Aug
27-
Sep
1
"... In virtually all respects, IFIP was a programming-oriented conference. Papers on hardware, circuit design, advanced components, etc., drew the smallest attendance while sessions on ALGOL, artificial intelligence, information retrieval were presented to capacity audiences. . . . The interest of Europe in ALGOL was exemplified by numerous signs accompanying equipment exhibits and, of course, in frequent conversations throughout the Congress. In Europe, FORTRAN is generally viewed as 'that other language' . . ."
- [132] 62 First meeting of IFIP Working Group 2.1 on ALGOL, in
Aug Munich. R. F. Clippinger reporting in *Datamation* their
28-30 plans to:
"a. to propose ALGOL 60 as an international standard,
b. to define I/O conventions for ALGOL 60,
c. to define an ALGOL 60 subset."

D. W. Hooper, President of the British Computer Society, in his annual report, published in the 62 Dec issue of the *Computer Bulletin*:

"... ALGOL 60 in its final international official version, including certain minor amendments, will be published in any country that wishes in this next few months. At one of the sessions the United States delegate stood up and publicly apologised for the lack of interest taken by America in ALGOL. . . . The United States are, of course, now full members of the IFIP Subcommittee which is now taking over ALGOL. . . . I expect the next full meeting will probably be about November to start on the revised edition of ALGOL which is ALGOL (60 + X) because we do not know the year. We have full American participation and support and this will pull ALGOL more into line with American thinking.

Outside official IFIP circles, it is my impression, and I would not for the moment put it any stronger than that, that in about 1969 or 1972 there will be the third edition of a standard language which will, if you like, overtake ALGOL, FORTRAN and all the lot. . . . There is at the moment strong international feeling that there must be one language, that ALGOL has served a purpose; it can still continue to serve a purpose, and I think it is certain that any future standard language can always look back to ALGOL as its honourable ancestor. . . ."

[133] 62 Third meeting of ISO/TC97/WG E on Programming
Oct Languages, in Paris.

9-13

IFIP invited to present a specification of ALGOL 60 (Rome version) and a proper subset for consideration as international standard programming languages. IFIP then submitted the official IFIP ALGOL and agreed that a subset specification would also be submitted, if and when completed.

The U.S. submitted a position paper, ISO/TC97/WG E (USA-19) 80:

"The recommendations below are submitted in anticipation of the possible proposal . . . that consideration be given to adoption of ALGOL 60 (Rome), recently approved as an IFIP official language, as an international standard language or ISO Recommendation. . . .

A. ISO/TC97/WG E should be concerned with ALGOL 60 (IFIP) as a potential programming language standard, and not merely as a publication language.

B. ALGOL 60 (IFIP) should not be considered acceptable as a Proposed Standard Programming Language without provision for or resolution of the following:

1. Input-output facility. . . .

2. A standard subset. . . .

3. . . . the five problem areas of ALGOL 60 (Rome) should be resolved by IFIP/WG 2.1. . . .

C. A means should be provided to determine whether or not an implementation satisfies the standard.

1. . . . a set of test programs, with a description of their behavior, to be included as part of any standard ALGOL. . . .

2. It is further recommended that WG E limit its language-measuring activity to the provision of test programs. . . .

D. The relationship between WG E and IFIP/WG 2.1 should be such that WG E as a standards processing authority will normally refer all technical or developmental problems and proposed solutions re ALGOL to IFIP/WG 2.1. . . .

. . . it is the hope of the USA that the general sense of the recommendations above will in any case be accepted and considered by the WG E group defining the language standardization procedure and program of work as essential elements thereof."

Working Group E accepted the IFIP ALGOL specification for consideration as a possible ISO Recommendation, and assigned it for study for the next meeting.

[134] 62 J. H. Wegstein to X3.4 and IFIP WG 2.1:

Dec
28

"On December 13, 1962, the BEMA Committee, X3.4 resolved that it is the USA position that ALGOL 60 (Rome) should not be considered as a standard without first dealing with input-output facilities and possibly even the settlement of the questions left by the Rome conference on ALGOL as well.

I believe that this does not represent the view of many people in the United States and other countries who are using ALGOL 60. . . ."

- [135] 63 U.S. position on IFIP ALGOL, Document ISO/TC97/
Jan SC5(USA-1)5:
7 This paper, emanating from ASA X3.4, informs ISO/TC97/SC5 that the U.S. is willing to consider as a standards proposal a version of ALGOL based on ALGOL 60 (Rome) with input-output facilities added, and/or the same for a subset of ALGOL 60 as developed by IFIP.
- [136] 63 N. Sanders and C. Fitzpatrick, in *Datamation*:
Jan "... The primary shortcoming of ALGOL as a computer language is its lack of a subroutine facility—a facility not required, of course, by a publication language. . . .
- The incorporation of the CALL statement changed the nature of a FORTRAN listing. No longer was it possible to read a FORTRAN program *per se* and understand it fully. The concept of remote compilation and, more seriously, remote *description* made it necessary for the reader of a FORTRAN program to have knowledge not contained in the listing itself. Consequently FORTRAN could have no claim to being a communication language and made no such claim. . . . As ALGOL is presently defined the language tail will wag the computing dog. Because of ALGOL's desire to communicate man to man it does not have, rightly, any subroutine facility. Consequently the whole philosophy of computer operations would have to change. No longer the library tape! . . .
- It would be worthwhile to consider breeding a FORTRAN hybrid which would be capable of string manipulation and which would use a stack for at least parameter transmission to subroutines, thus making it ALGOL-like internally and allowing it to compile itself. . . ."
- [137] 63 SHARE XX Session on the 7090 ALGOL Compiler:
Feb "About 150 people attended this session." A 21-page report was
27 distributed—"An Introduction to the SHARE ALGOL 60 Translator," by R. G. Franciotti of IBM.
- [138] 63 F. Jones, in *Datamation*:
Mar "... ALGOL, on the other hand, faced the *de facto* standard, FORTRAN, and the pragmatics of the situation were and are such that popularity is not in the cards for ALGOL—no computer user who has a large library of FORTRAN programs, or who has access to the huge collective FORTRAN library, can justify the cost of conversion to a system which most are not even sure is superior. . . ."

- [139] 63 Marjorie Lietzke (Manager, SHARE ALGOL Project) to
Apr the SHARE Membership:

1

"The SHARE Algol Project has reached a very important milestone. The first version of the SHARE Algol 60 Translator has been sent . . . for SDA distribution. . . .

We wish to emphasize that this is an experimental, not yet completely debugged, and in some respects not too efficient translator. However, it does implement most of ALGOL 60, and produces object code capable of giving correct answers on fairly complicated algorithms. . . .

For your convenience, as well as our own, we have integrated this first version of ALGOL with the FORTRAN II version 2 monitor so that the system tape is capable of running FORTRAN, FAP, and ALGOL. The libraries and operation are completely compatible. Later we plan to have ALGOL operating under IBSYS."

- [140] 63 Marjorie Lietzke (Manager, SHARE ALGOL Project) to
Apr Roy S. Dickson (Chairman, SHARE FORTRAN):

1

"I read, with considerable interest, your proposals for extensions to the FORTRAN IV language (SSD102, C-3179). A number of the items you mention have been implemented in the SHARE ALGOL 60 Translator. To mention a few:

1. Labels may be either numeric or alphabetic.
2. A statement label may be used as a parameter, thus permitting non-standard return from a subroutine.
3. The number of dimensions for a subscripted variable is not limited.
4. Array storage allocation may be completely dynamic, that is, all of it may be done at object time. There is no need for any dimensions to be fixed at compile time.
5. Subscript range checking is done at execution time, and subscripts may be positive or negative.
6. The loop control statement of Algol (*for* statement) may have positive or negative increments, either integer or floating point."

- [141] 63 Letter to the Editor of *Datamation*, from A. L. Cook:

Apr

" . . . It is true that many of these compilers (European ALGOL) do not include a subroutine facility as defined by Messrs. Sanders and Fitzpatrick; this is, however, a limitation of the *compiler* rather than the ALGOL *language*. There is no difficulty in providing a library tape of pre-compiled ALGOL procedures. These need be subject to no restriction on generality and may make free use of global variables. The procedures would be automatically found and inserted into the correct block-level (not necessarily the outer block) of the object program as a single procedure call directed at the compiler."

- [142] 63 J. W. Granholm, in *Datamation*:
 Apr "... Feb. 27th, in San Diego, Calif., the ALGOL Committee of the SHARE organization reported in open tutorial session. Gist of their report: ALGOL 60 is running on four 7090 installations—Rocketdyne ... General Atomic ... Oak Ridge National Laboratory ... and Marshall Space Flight Center. ... The master tapes ... are now available to any SHARE member. ...
 ALGOL, named by the Arabs, is a fixed star in the constellation Perseus. It was among the first of stars noted for its periodic variation in brightness, due to eclipse by its dark satellite. Its name, in Arabic, signifies 'The Demon'. On last Ash Wednesday in San Diego, ALGOL might have proven not only to be a demon, but to be a genie rising with astounding magic from the bottle where it had been securely corked by its critics."
- [143] 63 ASA Subcommittee X3.4, Programming Languages.
 Apr This reaffirmed the U.S. position of standardizing ALGOL on an
 25 international level rather than national, even should a national level need arise, which so far has not.
- [144] 63 Fourth Meeting of ISO/TC97/WG E, in Berlin, as reported
 Jun by H. Bromberg in 63 Aug issue of *Datamation*:
 5-7 "... The French ALGOL translation is currently in circulation in France for approval ... a straight translation which is to be used primarily for training purposes, and for promoting the implementation of ALGOL in French language countries. They recommend, however, that English words be used for programming purposes. The French standardization group has also prepared a draft proposal for standard hardware representation of ALGOL symbols. ...
 Germany reported presentation to the German standardization body of a first draft specification of an ALGOL 60 subset. This supersedes the ALCOR subset which had been previously considered. In addition, Germany is now considering a draft proposal to ISO/TC97/SC5 on representation of ALGOL symbols in five-channel tape and 80-column cards. Finally, they have prepared an English-German glossary of ALGOL Technical Terms for publication. ...
 The Italian National Activity Report stated that they are in the process of preparing a survey on programming languages and compilers. ... The Italian Standardization group is contributing to ALGOL ... through ECMA ... an Italian version of ALGOL 60 is being prepared.
 ... the Netherlands agree to all the points of the U.S. position on ALGOL except that they believe that changes should not be made to the revised ALGOL 60 language, but rather should be considered for the next ALGOL specification. ...
 Sweden reported that four ALGOL compilers ... are being constructed. Their standardization committee is also concerned with the problems of compatibility among the various ALGOL compilers.

The United Kingdom reported the establishment of a Programming Languages Technical Committee, DPE-13, under the British Standards Institution. . . . The United Kingdom section of the programming languages survey was updated and received by the U.S. Secretariat. . . .

The European Computer Manufacturers Association (ECMA) TC5 on ALGOL has been working on the preparation of an ALGOL subset which includes as many of the characteristics of the proposed IFIP subset as were known. . . .

. . . discussion of Subcommittee 5 resulted in unanimous approval of the following motion:

'SC5 received with great interest the IFIP ALGOL 60 revised report and deems it a significant contribution to ISO/TC97/SC5 standardization work. However, the committee feels that this document in its present form is incomplete in that standard input-output procedures and specification of a proper subset should be included. Therefore, SC5 invites IFIP to submit at its earliest convenience a more complete document.'

. . . the following resolution was unanimously approved:

'It is premature to decide today which choice we should make between ALGOL and FORTRAN due to the relative incompleteness of both documents presented to Subcommittee 5 and the fact that no criteria for evaluating a standard exist. It is therefore moved that the two condidate languages in the field of scientific programming be treated in parallel.'

The ALGOL Ad Hoc Working Group, under the chairmanship of William Heising of the USA reported consideration of the private Ingerman-Merner paper on ALGOL, which was presented as an example of current thinking in the United States. . . .

The Ad Hoc Working Group on FORTRAN, chaired by W. van der Poel of the Netherlands, presented the final report. . . ."

(It is of interest to note that Heising, who finally got together an IBM standard FORTRAN document to present to ASA, and van der Poel, who chairs the IFIP ALGOL Working Group, had ISO assignments which swapped the languages for which they were responsible. One can conclude correctly that this was deliberate and should pay off well.)

[145] 63 J. C. Boussard to the SHARE Secretary:

Jul "I am pleased to let you know that the computation department in
26 Grenoble has constructed a compiler for the Algol Language on IBM 7090-7044 computers.

The program, brought up for the first time at the Grenoble meeting (February 1963) translates Algol 60 instructions into FAP instructions directly performed by 7090/7094 and 7044 machines.

Since February 1963, this program was improved by being transferred upon an IBSYS-System-Tape, version 6, which allows us

from now to assemble and perform any number of ALGOL, FAP and FORTRAN programs, only parted at the input by "job" cards and a certain amount of control cards (see Fortran monitor).

During the following months, the same program will be added to the IJOB system, both upon 7090 and 7044 machines which will make it possible to use efficiently the IIBM assembler.

Our compiler is designed to accept any ALGOL program, with these few restrictions only:

- the input program must neither include a numerical label nor nested strings.
- all formal parameters must be specified and the type of every actual parameter must be identical to that of the formal corresponding parameter.
- all identifiers and labels written out in a switch declaration must be declared (defined for the labels) in the block where the declaration is located, or in a block outside.
- formal parameters specified as "label" cannot be called in by "value".

Other restrictions laid on input programs for the time being (recursive procedures, "arrays" called in by VALUE) are to be cancelled in the following next months.

A range of input-output procedures was defined for that compiler, they go from immediate input procedures to input-output procedures with specifications of FORTRAN formats. At last, all standard procedures advised by ALGOL committee on one hand, and all those which may be used in FORTRAN on the other hand, may be used in the input programs of the compiler.

The compiling method for this program: sequence of two passes of the input program (edition and generating), and wide-spread use of stacks make it possible to translate an ALGOL program into FAP instructions forming a program whose bulk and efficiency may be directly compared to those of FORTRAN II, version 3."

[146] 63 Working Conference on Mechanical Language Structures,
Aug in Princeton. Sponsored by the ACM, published in
14-16 *CACM*, 64 Feb.

[147] 63 Writing in *APIB* 18, E. W. Dijkstra reviews the GIER
Aug ALGOL manual.

Arguing against the need of subsetting ALGOL, he also notes that this is full ALGOL 60 except own arrays and arrays called by value, yet it was implemented for a machine with 1024 (40-bit) words of core store and 12800 of drum store, hardly extensive by today's measurements.

- [148] 63 A. S. Douglas, in *Datamation*, re the U.K. situation:
Aug "... Then, of course, one must these days have ALGOL (unless one is IBM). But ALGOL does not specify an input and output system much, and is not thought to be good for data processing. . . ."
- [149] 63 F. L. Alt succeeds R. F. Clippinger as Chairman of
Aug ASA X3.4.
- [150] 63 Third meeting of IFIP TC2, in Oslo.
Sep
9
- [151] 63 Second meeting of IFIP WG 2.1, in Delft. ECMA fur-
Sep nished a proposal for a subset of ALGOL.
10-13
- [152] 63 M. Lietzke to Manfred Paul, Mathematics Institute of
Sep Munich:
23 "Julien Green has informed me that your ALGOL Translator is now in the final check-out stage and that you are interested in having our SHARE ALGOL Project consider it as an alternative to the translator we have at present. Since our objective is to make the best possible Algol system available to SHARE members we would be most happy to consider your translator. . . ."
- [153] 63 M. Lietzke to Jean Claude Boussard, University of
Sep Grenoble:
30 "... I notice that you use French word delimiters; how difficult would it be to change the dictionary for your compiler to accept the standard English word delimiters? Do you have any provision for communicating with FAP or MAP assembled subroutines other than the built-in functions? . . ."
- [154] 63 J. C. Boussard to M. Lietzke:
Oct "... All the Standard Functions specified in Paragraph 3.2.4. of the
29 ALGOL 60 Report can be treated by the compiler. Input and output procedures are available and, in particular, FORTRAN-like Format Statements can be utilized by the programmer. The word delimiters or their abbreviations can be used arbitrarily in ENGLISH or in French. Notice that it is also valid to have a mixture of words in the two languages, as shown in the enclosed example. Some present restrictions of our compiler are as follows (1) ALGOL source programs should have less than 12,000 syntactical units, (2) the number of procedures is limited to 256, and, (3) the number of numerical constants should not exceed 2,000. As far as speed of compilation, we might add that it is comparable to that of FORTRAN II.

Among the restrictions to be observed in preparing source programs, the following are cited: (1) numerical labels are allowed, (2) own arrays with dynamic bounds are not permitted, and (3) all formal parameters must be specified and must be of the same type as actual corresponding parameters.

Turning to your last question, we do not have, at the present time, any means that permit the separate assembly of Algol and other languages. We are currently working on the problem of separate compilation with IJOB and IIBM.

[155] 63 A. P. Ershov to W. van der Poel, Chairman of IFIP
Nov WG 2, ALGOL:

4

"To my regret I shall not be able to attend at the second meeting of the WG 2. The main reason is that I have been received the official announcement too late (July 23, 1963) so I have no time to change my plans and to make necessary arrangements. . . .

3. Some news from the USSR:

- a. Two ALGOL translators for the M-20 computer are in an operation at present in the USSR. These are authorized by the Joint M-20 Users Commission attached to the Mathematical Institute, USSR AS. The first smaller translator for an ALGOL subset (no strings, no numerical labels, no recursive calls, no own, and some restrictions for procedure declarations and statements), consists of about 7000 instructions, multipass running, the speed of translation 1000-2000 operations per one source program symbol, a little optimization. The second translator which has been developed under guidance of Prof. Shura-Bura of Moscow University for full ALGOL minus dynamic arrays and numerical labels, consists of about 13,000 instructions, multipass running, the speed of translation 10-15 minutes per 1000 object instructions, some optimization. The ALPHA translator (Input Language without recursive calls and with some other minor restrictions) is now under experimental operation. It consists of 32,000 instructions, multipass running, the speed of translation about 5 minutes per 1000 instructions, careful optimization.
 - b. Bottenbruch's and Dijkstra's books on ALGOL 60 have been translated into Russian and are now in print. In addition, two or three original primers on ALGOL have been written and are in print too. An English translation of an extension of ALGOL 60 (Input Language) has been published in England and in the USA by the Academic Press.
 - c. There are 5 or 6 groups in the country wishing or beginning to develop translators for middle-size computers which are to be based on some ALGOL subsets. There are various opinions about the subsets but SMALGOL is in favor. . . .
8. I would like to make only one comment concerning possible discussion on ALGOL 60 at future meetings. I think it is necessary to separate problems of symbol manipulations from such points as complex arithmetics, matrix computations and so on.

I am sure that there should be an ALGOL-like, but separate language for string manipulations. I suppose September 1964 should be an appropriate date for the first discussion of the language."

- [156] 64 Resolutions of the 33rd Meeting of ASA X3.4:
Feb
20
- "2. That a Working Group† be established . . . to undertake standardization responsibility for ALGOL in the United States. . . .
 3. That the United States position on ALGOL at the ISO meetings include the position that an input-output system based on that of the Knuth report‡ (ISO/TC97/SC5(USA-90)40) be supported and endorsed as a part of standard ALGOL. . . .
 4. At this time X3.4 wished to place on record its recognition of the excellent results produced by Don Knuth and the members of his ACM group. They have produced a good I/O set of facilities for ALGOL promptly at a critical time in the progress of standardization of ALGOL. . . ."
- † X3.4.8, Chairman—J. Merner.
‡ In *CACM*, 64 May.

- [157] 64 Third meeting of IFIP WG 2.1., in Tutzing.
Mar
16-20
- The *ALGOL Bulletin* planned to be revived (No. 16 in May).

- [158] 64 R. F. Brockish to H. Bromberg, Chairman, Joint Users
Mar
17
- Group:
"At SHARE XXII in San Francisco, March 2-6, The SHARE Executive Board endorsed the following recommendation to ASA X3.4 from SHARE.

Recognizing (1) that ALGOL and FORTRAN are useful in closely related application areas and (2) that FORTRAN is still the more widely used of these two languages in the United States: X3.4 hereby instructs its delegation to the forthcoming ISO/TC97/SC5 meeting to support *no* action that would result in the consideration of an international standard ALGOL prior to equivalent consideration of an international standard FORTRAN.

Mr. Lynn Yarbrough of North American Aviation who is SHARE'S representative on X3.4 will present this recommendation to that group for acceptance.

SHARE's position in this matter is that FORTRAN is a widely used language that deserves equal attention when the question of an international standard computer language is considered by ISO. We feel that if ALGOL is considered without concern for FORTRAN and is declared a single standard, the chance of FORTRAN becoming a co-standard is remote. We feel that in the area of computer-independent languages for scientific applications, there is justification for both a standard ALGOL and a standard FORTRAN."

- [159] 64 R. F. Brockish to M. Lietzke, in SSD 119:
 Mar "I am writing on behalf of the SHARE Executive Board to convey to
 18 you our response to your recommendation that SHARE request IBM to assign one man to the maintenance of the ALGOL compiler. The SHARE Executive Board discussed your recommendation at length and does not consider that such a request would be in the best interest of SHARE. As you know by a vote of the general body at SHARE XVI in San Francisco, SHARE endorsed FORTRAN as the primary algebraic language and rescinded its request to IBM for an ALGOL compiler for SHARE machines. In keeping with the spirit of this resolution, although it in itself did not mention maintenance, the SHARE Executive Board feels that it should not request IBM to obligate itself to the maintenance of the SHARE developed ALGOL compiler.
 The Executive Board recognizes and thanks you for your enthusiastic efforts in developing the SHARE ALGOL compiler."
- [160] 64 Resolution of X3.4, in Washington:
 Apr "In view of the extensive development and preparatory work
 21 resultant from the initiative and request of TC97/SC5 and the USA as Secretariat, the USA urges SC5 to take action at its May 1964 meeting to enable a First Draft ISO Proposal Specification of ALGOL to be prepared for immediate circulation to SC5 under ISO Rules, and the USA will support such action."
- [161] 64 Fourth meeting of IFIP TC 2, in Liblice, Czechoslovakia
 May (near Prague):
 11 Proposals for IFIP SUBSET ALGOL 60 and input-output procedures were submitted by WG 2.1 and approved, both by TC 2 and the IFIP Council. This was in response to the ISO request. These proposals, labelled 'final issue—22 Apr 64', appear in ALGOL Bulletin 16, 64 May. It should be noted that the subset proposal derived mainly from the ECMALGOL, and ECMA standard (having previous input from ALCOR and SMALGOL).
 News item in 64 Jun issue of *Datamation*:
 "Moderation (and ALGOL) win out in Europe—A strongly worded attack on FORTRAN and IBM's new programming language—made at a recent IFIP meeting in Prague—has been toned down, we hear, and now constitutes a suggestion for cooperation between ALGOL Working Groups and NPL development representatives."
 ..."
- [162] 64 Announcement to SHARE ALGOL Mailing List:
 May "The SHARE ALGOL 60 Translator, MOD4, has been sent to SDA
 21 this week. This version of the translator shows a considerable increase in speed over previous releases. ...
 MOD4 will compile procedures separately, and will accept the standard ALCOR hardware representation with escape symbols, as well as our original reserved word hardware representation."

- [163] 64 Fifth meeting of ISO/TC97/SC5, in New York:
 May Major concentration led to the preparation of the *ISO Draft Proposal on the Algorithmic Language ALGOL*, prepared by an Ad Hoc Working Group on May 26-27. It included:
 25-28
- (1) A full ALGOL based upon the IFIP specification.
 - (2) A unique subset based upon the IFIP specification.
 - (3) An elementary level of I/O procedures based upon the IFIP specification.
 - (4) A level of I/O procedures based upon the Knuth report.
 - (5) An appendix containing the transliteration table between the ALGOL symbols and the ISO 6- and 7-bit code proposals.
- This Proposal was accepted for processing as an ISO proposal. A paper was prepared on "Criteria for Standardization of a Programming Language" (published in the *Computer Bulletin*, 65 Mar).
- [164] 64 Noted in the *ALGOL Bulletin* No. 17, ALGOL compilers for Atlas and the CDC 3600, and a note from J. H. Wegstein:
 Jul
- "There has been an overwhelming response to the algorithm reprint offer in the April 64 issue of the ACM Communications. . . . The speed with which requests have come in from several countries has been very encouraging as far as ALGOL interest is concerned. My supply of reprints was wiped out."
- [165] 64 From the Minutes of SHARE XXIII:
 Aug "Mr. L. Bolliet . . . described the ALGOL compiler which was developed under his supervision at the University of Grenoble. This compiler was written in the intersection of the 7090/94 and 7040/44 instruction sets. On the 7090 it operates under IBSYS and uses the FORTRAN II Version 3 System to assemble and load the object code. On the 7040/44 the compiler operates under IBJOB and translates to MAP.
 17-21
- The compiler has been submitted . . . for distribution."
 "The ALGOL project presented a report for the compiler on 7040/44 and will continue to support the 7090/94 compiler under its new manager" (John Whitney).
- [166] 64 E. L. Manderfield to the Editor of the *ALGOL Bulletin*:
 Aug "If you have any contact with any of the official European ALGOL-ers who have some influence, I would like to suggest that they proceed with ALGOL 6X because among the ranks of the American SMALGOLers there has been a mass desertion to 'NPL'. This is for two reasons, one is that ALGOL has never been a very popular language in America (partly because of the influence of the preponderance of IBMers, and partly because the ALGOL Maintenance Committee didn't make very many friends the way they operated); and the other is that NPL has adopted apparently the best features of the current programming languages. . . ."
 30

- [167] 64 News item in the *Computer Bulletin*:
 Sep "CPL is a programming language which has been developed jointly by members of the University Mathematical Laboratory, Cambridge, and the University of London Institute of Computer Science. . . . It is based on, and contains the concepts of ALGOL 60; in addition there are extended data descriptions. . . . However, CPL is not just another proposal for the extension of ALGOL 60, but has been designed from first principles, and has a logically coherent structure. . . ."
- [168] 64 One week course on Computational Linear Algebra and
 Sep Computer Programming in ALGOL, at the University of
 14-18 Manchester.
- [169] 64 Fourth meeting of IFIP WG 2.1, in Baden.
 Sep D. McIlroy, of the SHARE Advanced Language Development
 14-19 Committee, gave a presentation on NPL (New Programming Language). The meeting was devoted mainly to discussions of ALGOLs X and Y, X being a considerably extended and revised version, whereas Y is to be a completely new language with a rigorous definition in a metalanguage.
- [170] 64 Working Conference on Formal Language Description
 Sep Languages, under auspices of IFIP TC 2, Dr. H. Zemanek,
 15-18 Conference Chairman. Proceedings published by North-Holland, 1966.
- [171] 64 P. Naur, in the *ALGOL Bulletin* No. 18:
 Oct "In designing ALGOL 60 the assumption was implicitly made that a program written in a language common to many machines cannot take advantage of special features of any of them. . . . A better way of description is that ALGOL 60 prescribes one particular abstract machine. Particular implementations must then simulate this machine as best they can. . . . In passing it may be noted that the same kinds of difficulties are present in COBOL. In COBOL all data are basically expressed in terms of character strings. Simulating COBOL in a machine equipped with fast binary arithmetic is therefore excessively wasteful. . . ."
 (A proposal follows for a version of ALGOL with environment and data divisions.)
- [172] 64 News item in *Datamation*:
 Oct ". . . IBM is committing itself to the New Programming Language. Dr. Brooks said that COBOL and FORTRAN compilers for the System/360 were being provided 'principally for use with existing programs'. . . . Further, announced plans are for only two versions of COBOL . . . and two of FORTRAN . . . but four of NPL. . . ."

- [173] 64 Meeting of the ALCOR Group, in Bad Soden/Taunus, to
Oct consider the effects of ALGOL X on processor construc-
19-20 tion. Attendance by about 80 persons.
- [174] 64 Publication, in *Datamation*, by C. J. Shaw of an algorithm
Dec (in something like ALGOL) for singing the old Christmas
favorite, "A Partridge in a Pear Tree (APIAPT)". Good
fun.
- [175] 65 SHARE XXIV Session Report, by J. R. Whitney, Chair-
Mar man:
3 "The last rites of SHARE ALGOL were held in the Garden Room
East. In attendance were approximately 15-20 mourners, most of
them there out of curiosity, I suppose. Not a single tear was shed
when it was announced that SHARE ALGOL had become part of
history."
- [176] 65 Note in the *Computer Bulletin*:
Mar "Great interest has been shown in the proposed card-index scheme
for algorithms. No details of this scheme have yet been finalized, but
the editor would be glad to hear from any concern or individual
interested, particularly if they have suggestions as to how it might
best be implemented. . . ."
- [177] 65 IFIP 65 Congress, New York City.
May
24-29
- [178] 65 B. A. Galler to the Editor of *Datamation*:
May "Your editorial of March, 1965, strongly suggests that the manu-
facturers be handed the language development business 'without the
confusing influence of users who want every software system to
include their pet esoteric options which tend to reduce compiler
speed and efficiency.' I must remind you of some historical facts:
(1) If a group consisting largely of users hadn't come up with ALGOL
in the face of a practically standard FORTRAN, there would be
very little that is new in NPL.
(2) Some of us users were involved in showing the manufacturers
that it is possible to have both a decent language (not pet options)
and compiler speed.
(3) If users hadn't objected *violently* to early versions of NPL, you
would have found procedure-oriented languages set back some
years. (You have only to compare the final version of NPL with
versions 1 and 2 to see what I mean.) . . .
I note that ASA is now following the progress of NPL. How in the
world could anyone seriously consider NPL as a standard now when
not a single program has gone through a computer? . . . NPL

wasn't even announced until it had gone through six versions. It must surely be expected to go through another six before it settles down. . . .

What harm is there in watching it as a potential standard, I will be asked? Has anyone ever tried to make changes in something which is 'almost a standard'? And there will be those who are pleased if no changes can be made. But let us be forever grateful that FORTRAN I is not a standard now."

- [179] 65 A. d'Agapeyeff, in *Datamation*:
 May "The absence in Europe of a large vested interest in FORTRAN has led to a ready acceptance of the advantages of ALGOL as a language. It is the main vehicle for university teaching and is in widespread use particularly in Holland, Germany and Scandinavia. In Germany ALGOL or ALGOL-like compilers have been available for some six years, allowing an extensive body of experience to be built up.
- In Britain the progress of ALGOL has been more hesitant. . . . Furthermore, with one striking exception, it is only recently that really useful ALGOL compilers have been released. However, it is now the policy of two of the three main British manufacturers to support ALGOL, and it is backed by the majority of the universities. It would seem, therefore that, subject to the future impact of NPL, Britain will go along with the rest of Europe in favouring ALGOL."
- [180] 65 Fifth meeting of IFIP WG 2.1, in Princeton.
 May
 17-21
- [181] 65 Fifth meeting of IFIP TC 2, in New York.
 May
- [182] 65 News item in *Datamation*, "U.S.S.R.'s Evshov Speaks in
 Jul L.A.":
- "Primary programming effort in recent years has been in the development of ALPHA, an extended-ALGOL compiler (45K instructions, 20 passes), five years in the making. Designed for use on the M-20 (a 4K core machine which averages some 20,000 instructions/second), the compiler produces object programs at the rate of 150 three-address instructions/minute. 'It was very difficult,' Evshov said, but ALPHA has been in operation one year, and some 2000 programs have been translated, and the compiler has increased program production by two or three times. . . .
- Of other computer work in the Soviet Union, Evshov noted the announcement of plans for ALGEC, a language which will combine ALGOL and an economic (or business) language. . . ."

- [183] 65 ACM Programming Languages and Pragmatics Conference, in San Dimas.
Aug
8-12 Proceedings in *CACM*, 66 Mar.
- [184] 65 Sixth meeting of ISO/TC97/SC5, in Copenhagen.
Sep
6-10
- [185] 65 SHARE-JUG Conference on Programming Language Objectives of the Late 1960's, in Philadelphia, as reported in *Datamation*:
Oct
7-9
- "IBM, said William McClelland, does not see 'any need for any other major new procedure-oriented language development which is not a direct and essentially compatible extension of the existing languages.' . . . The ideal, he proffered, one IBM is working on, would be that 'manufacturers provide not compilers for languages themselves as a principal product, but a metalanguage compiler and expressions of standard forms of the appropriate language in that metalanguage. Users would then be able to tailor the language to their individual needs. . . .
- What about existing languages? Standards on two, FORTRAN and ALGOL, are (at time of writing) being voted upon at the ISO meeting in Japan. Will manufacturers support these standards? Attitudes, although positive, varied slightly. IBM gave an unequivocal *yes* for FORTRAN and COBOL, while UNIVAC noted that it would be 'guided by the needs' of its users. . . .
- A GSA spokesman on PL/I was quoted by a panelist as saying the 'government would have to protect its investment in COBOL and FORTRAN.'" (Elsewhere it was referenced that the 1964 inventory of government EDP equipment lists 5885 applications, 19% using FORTRAN as primary language, 3%—COBOL, and 1.8%—ALGOL.)
- [186] 65 ISO/TC97 Plenary meeting in Tokyo.
Oct
- The delegates of all national members approved the draft proposals, with the exception of the Netherlands, and the USSR, which abstained. The form was then changed to an ISO Recommendation, which will be prepared by an Editing Committee for circulation to ISO members for final approval. At the same time the levels of the language were increased from two to four—ALGOL 60, ECM-ALGOL plus recursion, ECMALGOL and the IFIP SUBSET. The transliteration tables will not be included, since no agreement could be found.
- [187] 65 News item in *ALGOL Bulletin* No. 21:
Nov
- "Telefunken have incorporated the I/O procedures proposed by the ACM Committee . . . into the ALGOL-System for the TR4 computer. . . . User's comments are very favourable. . . ."

- [188] 65 Sixth meeting of IFIP WG 2.1, in St. Pierre de Chartreuse.
 Oct The main topic of work continued to be ALGOL X.
 25-29
- [189] 65 Sixth meeting of IFIP TC 2, in Nice.
 Nov
 2
- [190] 66 46th meeting of X3.4, Common Programming Languages
 Mar in New York.
 11 Add up the plane fare and expenses for standardization work.
- [191] 66 Seventh meeting of IFIP TC 2, in London.
 Apr
 26 It was reported that the ISO Recommendation (ISO/TC97/SC2 Secretariat-26)102 had been edited and sent to AFNOR for translation into French. FORTRAN has already been through this process, but final action is delayed until completion of translation.
- [192] 66 Publication of "System 360 Operating System—ALGOL
 May Language"—IBM Form C28-6615-0.
- [193] 66 News brief in *Datamation*:
 Aug "IBM, whose giant software efforts for the 360 line are swallowing an estimated \$60 million in 1966, has taken on the development of another language compiler-ALGOL. Particularly aiming to meet the needs of the large ALGOL user group in Europe, the firm will deliver an F level, or 44K, compiler during third quarter 1967. It will meet the standard adopted by the European Computer Manufacturers Assn. and the International Federation of Information Processing. Said C. B. Rogers, director of systems marketing, "We believe System/360's PL/I and FORTRAN offer greater flexibility than ALGOL to scientific users, and we are encouraging conversion wherever it is practical."
- [194] 66 Training Course run by C.E.I.R., Arlington, Va., of
 Aug "ALGOL for FORTRAN Programmers".
 15-17

Algol References

COMMUNICATIONS OF THE ACM (Association for Computing Machinery)

Year	Vol.	No.	Page	
1958	1	8	3	Yershov, A. P., On Programming of Arithmetic Operations.
1958	1	8	12	Strong, J. <i>et al.</i> , The Problem of Programming Communication with Changing Machines, Part 1.

Year	Vol.	No.	Page	Paper
1958	1	9	9	Strong, J. <i>et al.</i> , The Problem of Programming Communication with Changing Machines, Part 2.
1958	1	10	5	Conway, M. E., Proposal for an UNCOL.
1958	1	12	8	Perlis, A. J. and Samelson, K., Preliminary Report, International Algebraic Language.
1959	2	2	6	Green, J., Possible Modifications to the International Algebraic Language.
1959	2	3	6	Wegstein, J., From Formulas to Computer-oriented Language.
1959	2	6	21	Williams, Jr., F. A., Handling Identifiers as Internal Symbols in Language Processors.
1959	2	9	19	Bemer, R. W., A Proposal for a Generalized Card Code for 256 Characters.
1959	2	9	24	ALGOL Subcommittee Report—Extensions.
1959	2	9	25	Green, J., Remarks on ALGOL and Symbol Manipulation.
1959	2	10	19	Kanner, H., An Algebraic Translator.
1959	2	10	25	Recommendations of the SHARE ALGOL Committee.
1959	2	12	14	Irons, E. T. and Acton, F. S., A Proposed Interpretation in ALGOL.
1960	3	2	76	Samelson, K. and Bauer, F. L., Sequential Formula Translation.
1960	3	3	170	Floyd, R. W., An Algorithm defining ALGOL Assignment Statements.
1960	3	4	211	Smith, J. W., Syntactic and Semantic Augments to ALGOL.
1960	3	4	213	Green, J., Symbol Manipulation in XTRAN.
1960	3	5	299	Naur, P. (Ed.), Report on the Algorithmic Language ALGOL 60.
1960	3	7	418	McIsaac, P., Combining ALGOL Statement Analysis with Validity Checking.
1960	3	8	463	Huskey, H. D., Halstead, M. H., McArthur, R., NELIAC, a Dialect of ALGOL.
1961	4	1	3	Huskey, H. D. and Wattenburg, W. H., A Basic Compiler for Arithmetic Expressions.
1961	4	1	10	Grau, A. A., Recursive Processes and ALGOL Translation.
1961	4	1	15	Bottenbruch, H., Use of Magnetic Tape for Data Storage in the ORACLE-ALGOL Translator.
1961	4	1	28	Arden, B. W., Galler, B. A., Graham, R. M., The Internal Organization of the MAD Translator.
1961	4	1	36	Evans, Jr., A., Perlis, A. J., Van Zoeren, H., The Use of Threaded Lists in Constructing a Combined ALGOL and Machine-like Assembly Processor.
1961	4	1	42	Floyd, R. W., An Algorithm for Coding Efficient Arithmetic Operations.
1961	4	1	51	Irons, E. T., A Syntax Directed Compiler for ALGOL 60
1961	4	1	55	Ingerman, P. Z., Thunks.
1961	4	1	59	Ingerman, P. Z., Dynamic Declarations.

Year	Vol.	No.	Page	Paper
1961	4	1	60	Sattley, K., Allocation of Storage for Arrays in ALGOL 60.
1961	4	1	65	Irons, E. T. and Feurzeig, W., Comments on the Implementation of Recursive Procedures and Blocks in ALGOL 60.
1961	4	1	70	Huskey, H. D. and Wattenburg, W. H., Compiling Techniques for Boolean Expressions and Conditional Statements in ALGOL 60.
1961	4	6	268	Knuth, D. E. and Merner, J. N., ALGOL 60 Confidential.
1961	4	9	393	Taylor, W., Turner, L., Waychoff, R., A Syntactical Chart of ALGOL 60.
1961	4	9	396	Rom, A. R. M., Manipulation of Algebraic Expressions.
1961	4	10	441	Jensen, J., Mondrup, P., Naur, P., A Storage Allocation Scheme for ALGOL 60.
1961	4	11	488	Strachey, C., and Wilkes, M. V., Some Proposals for Improving the Efficiency of ALGOL 60.
1961	4	11	499	SMALGOL 61.
1962	5	11	51	Algorithm Index, 1960-1961.
1962	5	1	54	Wegstein, J. H. and Youden, W. W., A String Language for Symbol Manipulation based on ALGOL 60.
1962	5	2	82	Schwarz, H. R., An Introduction to ALGOL.
1962	5	2	118	Forsythe, G. E., Von der Groeben, J., Toole, J. G., Vector-cardiographic Diagnosis with the Aid of ALGOL.
1962	5	3	145	Ledley, R. S. and Wilson, J. B., Automatic Programming Language Translation Through Syntactical Analysis.
1962	5	6	327	Rabinowitz, I. N., Report on the Algorithmic Language FORTRAN II.
1962	5	6	337	Thacher, Jr., H. C., A Redundancy Check for ALGOL Programs.
1962	5	7	376	Wegner, P., Communication Between Independently Translated Blocks.
1962	5	9	483	Floyd, R. W., On the Nonexistence of a Phrase Structure Grammar for ALGOL 60.
1962	5	10	505	Baecker, H. D., Implementing a Stack.
1962	5	11	547	Reiteration of ACM Policy Toward Standardization.
1963	6	1	1	Revised Report on the Algorithmic Language ALGOL 60.
1963	6	1	18	Supplement to the ALGOL 60 Report.
1963	6	1	20	Suggestions on ALGOL 60 (Rome) Issues.
1963	6	2	51	USA National Activity Report to ISO/TC97 Working Group E, Computers and Information Processing.
1963	6	3	77	Naur, P., Documentation Problems, ALGOL 60.
1963	6	3	93	Survey of Programming Languages and Processors.
1963	6	3	105	Brown, P. J., Note on the Proof of the Nonexistence of a Phrase Structure Grammar for ALGOL 60.
1963	6	4	159	Official Actions and Responses to ALGOL 60 as a Programming Language.

Year	Vol.	No.	Page	Paper
1963	6	4	169	Shoffner, M. G. and Brown, P. J., A Suggested Method of Making Fuller Use of Strings in ALGOL 60.
1963	6	6	294	Structures of Standards-Processing Organizations in the Computer Area.
1963	6	7	375	X3.4 forms ALGOL Task Group.
1963	6	8	451	Eickel, J., Paul, M., Bauer, F. L., Samelson, K., A Syntax Controlled Generator of Formal Language Processors.
1963	6	8	460	Kaupe, Jr., A. F., A Note on the Dangling <i>else</i> in ALGOL 60.
1963	6	9	502	USA National Activity Report to ISO/TC97, Subcommittee 5, Computers and Information Processing, 15 May 1963.
1963	6	9	544	An Open Letter to X3.4.2.
1963	6	9	547	Wirth, N., A Generalization of ALGOL.
1963	6	10	595	ECMA Subset of ALGOL 60.
1963	6	10	597	ALCOR Group Representation of ALGOL Symbols.
1963	6	12	721	Shaw, C. J., A Specification of JOVIAL.
1964	7	1	15	Forsythe, G. E., Revised Algorithms Policy.
1964	7	1	16	Garwick, J. V., GARGOYLE, a Language for Compiler Writing.
1964	7	2	52	Rose, G. F., An Extension of ALGOL-like Languages.
1964	7	2	62	Floyd, R. W., Bounded Context Syntactic Analysis.
1964	7	2	67	Irons, E. T., "Structural Connections" in Formal Languages.
1964	7	2	80	Iverson, K. E., Formalism in Programming Languages.
1964	7	2	89	Perlis, A. J., A Format Language.
1964	7	2	119	Brooker, R. A., A Programming Package for Some General Modes of Arithmetic.
1964	7	2	127	Perlis, A. J. and Iturriaga, R., An Extension to ALGOL for Manipulating Formulae.
1964	7	2	131	Ross, D. T., On Context and Ambiguity in Parsing.
1964	7	3	146	Algorithms Subject Index 1960-1963.
1964	7	3	189	Corrigenda: "ALCOR Group Representations of ALGOL Symbols".
1964	7	5	273	—, A Proposal for Input-Output Conventions in ALGOL 60—A Report of the Subcommittee on ALGOL of the ACM Programming Languages Committee.
1964	7	5	283	ASA X3.4 Meeting No. 33.
1964	7	5	288	Shaw, C. J., On Declaring Arbitrarily Coded Alphabets.
1964	7	5	297	Revised Algorithms Policy, May 1964.
1964	7	7	422	Garwick, J. V., Remark on Further Generalization of ALGOL.
1964	7	8	475	Lietzke, M. P., A Method of Syntax-Checking ALGOL 60.
1964	7	10	587	Wilkes, M. V., Constraint-Type Statements in Programming Languages.

Year	Vol.	No.	Page	Paper
1964	7	10	588	Iverson, K. E., A Method of Syntax Specification.
1964	7	10	626	Report on SUBSET ALGOL 60 (IFIP).
1964	7	12	703	Index by Subject to Algorithms, 1964.
1964	7	12	734	Petrone, L. and Vandoni, C. E., Integer and Signed Constants in ALGOL.
1964	7	12	735	Knuth, D. E., Backus Normal Form vs. Backus Naur Form.
1965	8	2	89	Landin, P. J., A Correspondence Between ALGOL 60 and Church's Lambda-Notation, Part I.
1965	8	3	147	Johnston, J. B., A Class of Unambiguous Computer Languages.
1965	8	3	158	Landin, P. J., A Correspondence Between ALGOL 60 and Church's Lambda Notation, Part II.
1965	8	3	167	Zaremba, W. A., On ALGOL I/O Conventions.
1965	8	3	200	Petrick, S. R., More on Backus Normal Form.
1965	8	5	275	Forsythe, G. E. and Wirth, N., Automatic Grading Programs.
1965	8	5	304	Burkhardt, W. H., Metalanguage and Syntax Specification.
1965	8	6	349	Galler, B. A. and Fischer, M. J., The Iteration Element.
1965	8	6	378	Weil, Jr., R. L., Testing the Understanding of the Difference Between Call by Name and Call by Value in ALGOL 60.
1965	8	7	427	Kanner, H., Kosinski, P., Robinson, C. L., The Structure of Yet Another ALGOL Compiler.
1965	8	8	496	Gries, D., Paul, M., Wiehle, H. R., Some Techniques Used in the ALCOR ILLINOIS 7090.
1965	8	11	671	Naur, P., The Performance of a System for Automatic Segmentation of Programs within an ALGOL Compiler (GIER ALGOL).
1965	8	12	786	Anderson, J. P., Program Structures for Parallel Processing.
1965	8	12	791	Index by Subject to Algorithms, 1965.
1966	9	1	13	Wirth, N. and Weber, H., EULER: A Generalization of ALGOL, and its Formal Definition, Part I.
1966	9	2	72	Parnas, D. L., A Language for Describing the Functions of Synchronous Systems.
1966	9	2	89	Wirth, N. and Weber, H., EULER: A Generalization of ALGOL, and its Formal Definition, Part II.
1966	9	3	137	Forsythe, G. E., Welcoming Remarks to the ACM Programming Languages and Pragmatics Conference.
1966	9	3	139	Zemanek, H., Semiotics and Programming Languages.
1966	9	3	143	Dennis, J. B. and Van Horn, E. C., Programming Semantics for Multiprogrammed Computations.
1966	9	3	157	Landin, P. J., The Next 700 Programming Languages.
1966	9	3	176	Naur, P., Program Translation Viewed as a General Data Processing Problem.

Year	Vol.	No.	Page	Paper
1966	9	3	179	Boussard, J. C., An ALGOL Compiler: Construction and Use in Relation to an Elaborate Operating System.
1966	9	4	255	A Forum on Algorithms (Perlis, Forsythe, Herriot, Engel, Ondis).
1966	9	4	267	Carr III, J. W., Weiland, J., A Nonrecursive Method of Syntax Specification.
1966	9	5	320	Wirth, Niklaus, A Note on "Program Structures for Parallel Processing".
1966	9	5	321	Knuth, D. E., Additional Comments on a Problem in Concurrent Programming Control.
1966	9	6	413	Wirth, N., and Hoare, C. A. R., A Contribution to the Development of ALGOL.
1966	9	8	549	Perlis, A., Iturriaga, R. and Standish, T. A., A Definition of Formula ALGOL.
1966	9	9	671	Dahl, O.-J. and Nygaard, K., SIMULA—An ALGOL-Based Simulation Language.
1966	9	9	679	Abrahams, P. W., A Final Solution to the Dangling <i>else</i> of ALGOL 60 and Related Languages.
1967	10	3	137	Ingerman, P. Z., "Panini-Backus Form" Suggested.
1967	10	3	172	Von Sydow, L., Computer Typesetting of ALGOL.
1967	10	4	204	Galler, B. A. and Perlis, A. J., A Proposal for Definitions in ALGOL.

JOURNAL OF THE ACM (Association for Computing Machinery)

Year	Vol.	No.	Page	Paper
1962	9	2	161	Bottenbruch, H., Structure and Use of ALGOL 60.
1962	9	3	350	Ginsburg, S. and Rice, H. G., Two Families of Languages Related to ALGOL.
1962	9	4	480	Grau, A. A., A Translator-oriented Symbolic Programming Language.
1963	10	1	29	Ginsburg, S. and Rose, G. F., Some Recursively Unsolvable Problems in ALGOL-like Languages.
1964	11	2	159	Randell, B. and Russell, L. J., Single-Scan Techniques for the Translation of Arithmetic Expressions in ALGOL 60.
1966	13	1	17	Evshov, A. P., ALPHA—an Automatic Programming System of High Efficiency.
1967	14	1	1	Perlis, Alan J., The Synthesis of Algorithmic Systems.

DATAMATION

Year	Issue	Page	Paper
1960	Sep/O	46	Flores, I., An Explanation of ALGOL 60, Part 1.
1960	Nov/D	65	Flores, I., An Explanation of ALGOL 60, Part 2.
1961	Sep	24	Wegstein, J. H., ALGOL 60,—A Status Report.
1961	Oct	41	—, ALGOL: a critical profile (RAND Symposium, Part 2).

Year	Issue	Page	Paper
1961	Nov	27	Utman, R. E., Language Standards . . . A Status Report.
1961	Nov	46	Shaw, C. J., A Programmer's Look at JOVIAL.
1961	Dec	24	Forest, R., BALGOL at Stanford.
1961	Dec	29	McCracken, D. D., Basic ALGOL.
1962	Feb	32	A Game to Counter Compileritis (Burroughs Corp.).
1962	Apr	88	McMahon, J. T., ALGOL vs. FORTRAN.
1962	May	34	Shaw, C. J., The Language Proliferation.
1962	May	44	McCracken, D. D., A New Home for ALGOL.
1962	Jun	33	Balch, B. and Gallie, T., ALGOL at Duke.
1962	Aug	25	Cantrell, H. N., Where are Compiler Languages Going?
1962	Oct	25	—, The RAND Symposium: 1962, Part 1.
1962	Nov	23	—, The RAND Symposium: 1962, Part 2.
1963	Jan	30	Sanders, N. and Fitzpatrick, C., ALGOL and FORTRAN Revisited.
1963	Apr	23	Editorial: Angels, Pins and Language Standards.
1963	Apr	26	Clippinger, R. F., Progress in Language Standards.
1963	Apr	28	Granholm, J. W., ALGOL on the 7090.
1963	Aug	41	Bromberg, H., Standardization of Programming Languages.
1964	Jul	31	McCracken, D. D., The New Programming Language.
1964	Dec	28	Shaw, C. J., that old favorite, Apiapt, A Christmastime Algorithm.
1965	May	31	d'Agapeyeff, A., Software in Europe.
1965	Jul	99	—, USSR's Evshov Speaks in L.A.
1965	Nov	141	—, Language in the Sixties.

THE COMPUTER JOURNAL
(of the British Computer Society)

Year	Vol.	No.	Page	Paper
1959	2	3	110	Gill, S., Current Theory and Practice of Automatic Programming.
1960	2	4	151	Gill, S., ALGOL Conference in Paris.
1960	3	2	67	Woodger, M., An Introduction to ALGOL 60.
1961	4	1	10	Huskey, H. D., Compiling Techniques for Algebraic Expressions.
1962	4	4	292	Hockney, R. W., ABS12 ALGOL, an Extension to ALGOL 60 for Industrial Use.
1962	5	2	125	Dijkstra, E. W., Operating Experience with ALGOL 60.
1962	5	2	127	Hoare, C. A. R., Report on the Elliott ALGOL Translator.
1962	5	2	130	Duncan, F. G., Implementation of ALGOL 60 for the English Electric KDF9.
1962	5	3	210	Hamblin, C. L., Translation to and from Polish Notation.
1963	5	4	332	Watt, J. M., The Realization of ALGOL Procedures and Designational Expressions.

Year	Vol.	No.	Page	Paper
1963	5	4	338	Gerard, J. M. and Sambles, A., A Hardware Representation for ALGOL 60 Using Creed Teleprinter Equipment.
1963	5	4	341	Duncan, F. G., Input and Output for ALGOL 60 on KDF9.
1963	5	4	345	Hoare, C. A. R., The Elliott ALGOL Input-Output System.
1963	5	4	349	Naur, P. (Editor), Revised Report on the Algorithmic Language ALGOL 60.
1963	6	1	50	Higman, B., What Everybody Should Know About ALGOL.
1964	6	4	336	Ryder, K. L., Note on an ALGOL 60 Compiler for Pegasus I.
1964	7	1	24	Pullin, D., A FORTRAN to ALGOL Translator.
1964	7	1	28	Parker-Rhodes, A. F., The Communication of Algorithms.
1965	8	1	21	Samet, P. A., The Efficient Administration of Blocks in ALGOL.
1965	8	2	113	Barnes, J. G. P., A KDF9 ALGOL List-processing Scheme.
1966	8	3	167	LITHP—An ALGOL Processor.

THE COMPUTER BULLETIN (of the British Computer Society)

Year	Vol.	No.	Page	Paper
1958	2	2	24	—, Automatic Coding by FORTRAN.
1959	2	6	81	—, Zurich Conference on Algorithmic Language.
1959	3	1	9	—, Towards a Common Programming Language.
1959	3	3	53	Wilkes, M. V., International Conference on Information Processing.
1959	3	3	64	—, Towards a Common Programming Language (2).
1960	3	5	87	—, Towards a Common Programming Language (3).
1960	4	1	18	—, Towards a Common Programming Language (4).
1961	4	4	127	Bemer, R. W., Survey of Modern Programming Techniques.
1962	6	2	47	Kilner, D., Automatic Programming Languages for Business and Science.
1964	7	4	107	Pearcey, T., Aspects of the Philosophy of Computer Programming.
1964	8	2	66	—, Algorithm Supplement.
1964	8	3	108	—, Algorithm Supplement.
1965	8	4	146	—, Algorithm Supplement.
1965	9	1	18	—, Algorithm Supplement.
1965	9	2	56	—, Algorithm Supplement.
1965	9	3	104	—, Algorithm Supplement.
1965	9	4	115	Programming in ALGOL (a review).

THE ALGOL BULLETIN

Issue	Date	Page	Paper
16	64 May	14	Dijkstra, E. W., A Simple Mechanism Modelling Some Features of ALGOL 60.
16	64 May	24	Duncan F. G. and van Wijngaarden, A., Cleaning Up ALGOL 60.
18	64 Oct	26	Naur, P., Proposals for a New Language.
22	66 Feb	28	Woodger, M., ALGOL X, Note on the Proposed Successor to ALGOL 60.

(Note: Only these few papers are listed here. The balance are mainly in the form of correspondence or have been published subsequently elsewhere.)

ELEKTRONISCHE RECHENANLAGEN

Year	Vol.	No.	Page	Paper
1959	1		72	Zemenek, H., Die algorithmische Formelsprache ALGOL.
1959	1		176	Samelson, K. and Bauer, F. L., Sequentielle Formelübersetzung (see also <i>CACM</i> 3, 1960).
1961	3		206	Baumann, R., ALGOL—Manual der ALCOR-GRUPPE, Part I.
1961	3		259	Baumann, R., ALGOL—Manual der ALCOR-GRUPPE, Part II.
1962	4	2	71	Baumann, R., ALGOL—Manual der ALCOR-GRUPPE, Part III.
1963	5	2	77	ALGOL Dictionary.
1965	7	5	239	Zemanek, H., Alphabets and Codes, 1965.
1966	8	2	81	Busse, H. G., A Possible Extension of ALGOL.

ELEKTRONISCHE DATENVERARBEITUNG

Year	Vol.	No.	Page	Paper
1964	6	6	233	Kruseman Aretz, F. E. J., ALGOL 60 Translation for Everybody.
1964	6	6	248	Schuff, H. K., Bemerkungen zu ALGOL 60.
1966	8	2	49	Muller-Merbach, H., Die Lösung des Transportproblems auf Rechenautomaten—ein ALGOL-Programm.
1967	9	1	3	Wieland, H., Speicherzuweisung für Variable in ALGOL Objektprogrammen.
1967	9	2	89	Knussmann, R., ALGOL-Rechenprogramme statistischer Standardverfahren.
1967	9	3	101	Schrader, K.-H., Eine Sprache und ein ALGOL-Programmsystem für Probleme der Mechanik der Systeme (MESY).

COMPUTER APPLICATIONS SYMPOSIUM (Armour Research Foundation, Chicago, now Illinois Institute of Technology Research Inst.)

Year	Page	Paper
1957	107	Bemer, R. W., The Status of Automatic Programming for Scientific Problems.
1959	112	Katz, C., The International Algebraic Language and the Future of Programming.
1960	154	Herriot, J. G., Some Observations on ALGOL in Use (Burroughs 220).
1961	115	Naur, P., The Progress of ALGOL in Europe.
1962	176	Clippinger, R. F., Data Processing Standards.
1962	204	Bemer, R. W., An International Movement in Programming Languages.

JOURNAL OF DATA MANAGEMENT

Year	Vol.	No.	Page	Paper
1966		8	56	New Program Supports ALGOL 360 Converts.

COMPUTERS AND AUTOMATION

Year	Vol.	No.	Page	Paper
1962	11	11	17	Clippinger, R. F., ALGOL—A Simple Explanation.
1962	11	12	8	Knuth, D. E., A History of Writing Compilers.
1964	13	11	32	Alt, F. L., The Standardization of Programming Languages.
1965	14	2	12	Chapin, N., What Choice of Programming Languages?
1965	14	2	15	Schwartz, J. I., Comparing Programming Languages.

NORDISK TIDSKRIFT FOR INFORMATIONS-BEHANDLING (BIT)—
(Danish Publication)

Year	Vol.	No.	Page	Paper
1961	1	1	38	Jensen, J. and Naur, P., An Implementation of ALGOL 60 Procedures.
1961	1	2	89	Jensen, J., Mondrup, P., Naur, P., A Storage Allocation Scheme for ALGOL 60.
1962	2	1	7	Dahl, O-J, Remarks on the Use of Symbols in ALGOL.
1962	2	3	137	Dahlstrand, I., A Half Year's Experience with the Facit-ALGOL I Compiler.
1962	2	4	232	Wynn, P., An Arsenal of ALGOL Procedures for Complex Arithmetic.
1963	3	2	124	Naur, P., The Design of the GIER ALGOL Compiler, Part I.

Year	Vol.	No.	Page	Paper
1963	3	3	145	Naur, P., The Design of the GIER ALGOL Compiler, Part II.
1964	4	2	115	Naur, P., Using Machine Code Within an ALGOL System.
1964	4	4	162	Langefors, B., ALGOL-GENIUS, a Programming Language for General Data Processing.
1964	4	3	177	Naur, P., Automatic Grading of Students' ALGOL Programming.
1965	5	2	85	Duncan, F. G., Possibilities for Refining an Object Program Compiled with an ALGOL Translator.
1965	5	3	151	Naur, P., Checking of Operand Types in ALGOL Compilers.
1965	5	4	235	Jensen, J., Generation of Machine Code in ALGOL Compilers.
1966	6	4	332	Tienari, M. and Suokonautio, V., A Set of Procedures Making Real Arithmetic of Unlimited Accuracy Possible Within ALGOL 60.

ANNUAL REVIEW IN AUTOMATIC PROGRAMMING
(Pergamon Press, Oxford, New York, 4 volumes)

Year	Vol.	Page	Paper
1959	1	268	Preliminary Report of ACM-GAMM Committee on an International Algebraic Language.
1961	2	67	Rutishauser, H., Interference with an ALGOL Procedure.
1962	3	1	Woodger, M., The Description of Computing Processes. Some Observations on Automatic Programming and ALGOL 60
		17	van Wijngaarden, A., Generalized ALGOL.
		27	Dijkstra, E. W., On the Design of Machine Independent Programming Languages.
		43	Rutishauser, H., The Use of Recursive Procedures in ALGOL 60.
		53	Shaw, C. J., JOVIAL, A Programming Language for Real-time Command Systems.
		121	Higman, B., Towards an ALGOL Translator.
		163	Hawkins, E. N. and Huxtable, D. H. R., A Multi-pass Translation Scheme for ALGOL 60.
		207	Irons, E. T., The Structure and Use of the Syntax-directed Compiler.
		329	Dijkstra, E. W., An ALGOL 60 Translator for the X1.
		347	Dijkstra, E. W., Making a Translator for ALGOL 60.
1964	4	1	Wilkes, M. V., An Experiment with a Self-compiling Compiler for a Simple List-Processing Language.
		49	Naur, P., The Design of the GIER ALGOL Compiler.
		87	Evans, Jr., A., An ALGOL 60 Compiler.
		167	Marsh, D. G., JOVIAL in CLASS.
		217	Revised Report on the Algorithmic Language ALGOL 60.

- 1959 PROCEEDINGS, INTERNATIONAL CONFERENCE ON INFORMATION PROCESSING (Paris, 1959 June 15-20, UNESCO, Verlag Oldenbourg, Munich, 1960)

Page	Paper
120	Bauer, F. L. and Samelson, K., The Problem of a Common Language, Especially for Scientific Numerical Work.
125	Backus, J. W., The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference.
132	Poyen, J. and Vauquois, B., Suggestions for a Universal Language (in French).
152	Symposium on Automatic Programming: (3) Huskey, H. D., A Variation of ALGOL. (5) Bauer, F. L. and Samelson, K., The Cellar Principle for Formula Translation.

- 1963 PROCEEDINGS, INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING (Munich, 1962 Aug 27-Sep 1, North Holland Publishing Co., Amsterdam)

Page	Paper
487	Samelson, K., Programming Languages and their Processing.
493	Paul, M., ALGOL 60 Processors and a Processor Generator.
498	Keese, Jr., W. M. and Huskey, H. D., An Algorithm for the Translation of ALGOL Statements.
503	Denison, S. J. M., A Proposed ALGOL 60 Matrix Scheme.
509	Lombardi, L. A., On Table Operating Algorithms.
513	Symposium of Languages for Processor Construction.
518	Symposium on Programming Languages.
524	Panel on Techniques for Processor Construction.
535	Dijkstra, E. W., Some Meditations on Advanced Programming.
556	Lucas, P., Requirements on a Language for Logical Data Processing.

- 1965 PROCEEDINGS OF IFIP CONGRESS 65
(Spartan Books, Washington, D.C., 648 pp.)

Vol. 1

Page	Paper
195	Naur, P., The Place of Programming in a World of Problems, Tools and People.
201	Gill, S., The Changing Basis of Programming.
213	Dijkstra, E. W., Programming Considered as a Human Activity.
223	Caracciolo di Forino, A., Linguistic Problems in Programming Theory.

Page	Paper
314	van der Poel, W. L., Recent Developments in the Construction of a new ALGOL.
315	Wirth, N. and Weber, H., EULER: A Generalization of ALGOL and its Formal Description.
316	Hauer, H. J., SYNALGOL—An Algorithm Language Capable of Growing.
438	Zemanek, H., IFIP Working Conference on Formal Language Description Languages.
438	Landin, P. J., An Abstract Machine for Designers of Computing Languages.
454	Perlis, A. J. and Standish, T. A., Formula ALGOL.
456	Engeli, M. E., Formal Manipulation of Algebraic Expressions with an Algorithmic Language.
458	Strachey, C., The Problems of Incorporating List-Processing in a General Purpose Programming Language.
554	Nygaard, K. and Dahl, O.-J., SIMULA—A Language for Describing Discrete Event Systems.
590	Petrone, L., Syntactic Mappings of Context-Free Languages.
620	Naur, P., Organizing the Use of Multi-Level Stores.
622	Evshov, A. P., ALPHA—An Automatic Programming System of High Efficiency.

1966 PROCEEDINGS OF THE IFIP WORKING CONFERENCE ON FORMAL LANGUAGE DESCRIPTION LANGUAGES (T. B. Steel, Jr. (Ed.), North Holland Publishing Co., Amsterdam, 1966)

Page	Paper
1	McCarthy, J., A Formal Description of a Subset of ALGOL.
13	van Wijngaarden, A., Recursive Definition of Syntax and Semantics.
25	Steel, Jr., T. B., A Formalization of Semantics for Programming Language Description.
76	Culik, K., Well-translatable Grammars and ALGOL-like Languages.
86	Ginsburg, S., A Survey of ALGOL-like and Context-free Language Theory.
139	Garwick, J. V., The Definition of Programming Languages by their Compilers.
148	Nivat, M. and Nolin, N., Contribution to the Definition of ALGOL Semantics.
221	Ingerman, P. Z., The Parameterization of the Translation Process.
249	Gorn, S., Language Naming Languages in Prefix Form.
266	Landin, P. J., A Formal Description of ALGOL 60.
295	Duncan, F. G., Our Ultimate Metalanguage.

1962 SYMBOLIC LANGUAGES IN DATA PROCESSING
(Gordon and Breach, New York, London, 849 pp.)

Page	Paper
23	Ingerman, P. Z., A Translation Technique for Languages Whose Syntax is Expressible in Backus Normal Form.
65	Paul, M., A General Processor for Certain Formal Languages.
75	Culik, K., Formal Structure of ALGOL and Simplification of its Description.
207	Samelson, K. and Bauer, F. L., The ALCOR Project.
219	Huskey, H. D., Machine Independence in Compiling.
229	van der Poel, W. L., The Construction of an ALGOL Translator for a Small Computer.
237	Dijkstra, E. W., An Attempt to Unify the Constituent Concepts of Serial Program Execution.
253	Kiyono, T. and Nagao, M., Comments on the ALGOL System for the Small and Medium Size Computers.
263	Palermo, G. and Pacelli, M., Sequential Translation of a Problem-Oriented Programming Language.
317	Piccifluoco, U. and Pacelli, M., Non-Dynamic Aspects of Recursive Programming.
325	Wohlfahrt, K., On Static and Dynamic Treatment of Types in ALGOL Translators.
331	Hill, U., Langmaack, H., Schwarz, H. R., Seegmuller, G., Efficient Handling of Subscripted Variables in ALGOL 60 Compilers.
341	Dolotta, T. A., A Method of Editing a Program in Symbolic Language.
385	Naur, P., The Basic Philosophy Concepts, and Features of ALGOL.
391	Woodger, M., The Description of Computing Processes. Some Observations on Automatic Programming and ALGOL 60.
409	van Wijngaarden, A., Generalized ALGOL.
421	Moriguti, S., A Family of Symbolic Input Languages and an ALGOL Compiler.
439	Pacelli, M., Gavioli, D., Palermo, G., Piccifluoco, U., PALGO, an Algorithmic Language and its Translator for Olivetti ELEA 6001.
449	Savastano, G. and Fadini, B., The Algebraic Compilers for Bendix G-20 Computing System.
473	Bosset, L., MAGE, A Language Derived from ALGOL Adapted to Small Machines.
481	Schwartz, J. I., JOVIAL, A General Algorithmic Language.
495	Katz, C., GECOM, the General Compiler.
501	Balke, K. G. and Carter, G. L., The COLASL Automatic Coding Language.
539	Mazurkiewicz, A., Compiler-Interpreter for Using in Numerical Oriented Languages Translation.

ALGOL (OR CLOSELY RELATED VARIANTS) BOOKS

- 1962 Dijkstra, E. W., *A Primer of ALGOL 60 Programming*, Academic Press, London, 114 pp.
- 1962 McCracken, D. D., *A Guide to ALGOL Programming*, Wiley, New York, 106 pp.
- 1962 Halstead, M. H., *Machine Independent Computer Programming*, Spartan Books, Washington, D.C., 267 pp. (NELIAC).
- 1962 Galler, B. A., *The Language of Computers*, McGraw-Hill, 244 pp. (MAD).
- 1963 Wooldridge, R. and Ratcliffe, J. F., *An Introduction to ALGOL Programming*, The English Universities Press, London, 131 pp.
- 1963 Evshov, A. P., Kozhukhin, G. I., Voloshin, U. M., *Input Language for Automatic Programming Systems*, Academic Press, London, 70 pp.
- 1963 Güntsch, F. R., *Einführung in die Programmierung Digitaler Rechenautomaten*, Verlag Walther de Gruyter, Berlin, 388 pp.
- 1964 Bolliet, L., Gastinel, N., Laurent, P. J., *Un Nouveau Language Scientifique: ALGOL: Manuel Pratique*, Hermann, Paris, 196 pp.
- 1964 Randell, B. and Russell, L. J., *ALGOL 60 Implementation, The Translation and Use of ALGOL 60 Programs on a Computer*, Academic Press, 418 pp.
- 1964 Baumann, R., Feliciano, M., Bauer, F. L. and Samelson, K., *Introduction to ALGOL*, Prentice-Hall International, London, 142 pp.
- 1964 Reeves, C. M. and Wells, M., *A Course on Programming in ALGOL 60*, Chapman and Hall, London, 82 pp.
- 1964 Anderson, C., *An Introduction to ALGOL 60*, Addison-Wesley, 57 pp.
- 1964 Math. Laboratory, Royal Radar Establishment, *Programming in ALGOL 60*, 29 pp.
- 1964 Nickel, K., *ALGOL-Praktikum: Eine Einführung in das Programmieren*, Karlsruhe, Braun, 220 pp.
- 1965 Nicol, K., *Elementary Programming and ALGOL*, McGraw-Hill, 147 pp.
- 1965 Bauer, F. L., Heinhold, J., Samelson, K., Sauer, R., *Moderne Rechenanlagen: Eine Einführung*, Stuttgart, Teubner, 357 pp.
- 1965 Hawgood, J., *Numerical Methods in ALGOL*, McGraw-Hill, 178 pp.
- 1965 Arsac, J., Lentin, A., Nivat, M., Nolin, L., *ALGOL: Theorie et Pratique*, Gauthier-Villars, Paris, 204 pp.
- 1965 Broise, P., *Le Langage ALGOL; Applications à des Problèmes de Recherche Opérationnelle*, Dunod, Paris, 99 pp.
- 1965 Ekman, T. and Fröberg, C.-E., *Introduction to ALGOL Programming*, Lund, 1965 and Oxford University Press, 123 pp.
- 1966 Schaeffler, G. F., *A Course in ALGOL Programming*, MacMillan, London 192 pp.
- 1966 Kerner, I. and Zielke, *Einführung in . . . ALGOL*, Teubner, Leipzig, 283 pp.
- 1966 Marcovitz, A. B. and Schweppe, E. J., *An Introduction to Algorithmic Methods Using the MAD Language*, MacMillan, New York, 433 pp.
- 1966 Ingerman, P. Z., *A Syntax-oriented Translator*, Academic Press, New York, 131 pp.
- 1966 Herschel, R., *Anleitung zum praktischen Gebrauch von ALGOL*, R. Oldenbourg Verlag, München, 162 pp.
- 1967 Lecht, C. P., *The Programmer's ALGOL*, McGraw-Hill, New York, 251 pp.
- 1967 Higman, B., *A Comparative Study of Programming Languages*, Macdonald, London, 164 pp.

VARIOUS OTHER PAPERS ON ALGOL

- Bottenbruch, H., Übersetzung von algorithmischen Formelsprachen in die Programmiersprachen von Rechenmaschinen, *Zeitschrift math. Logik Grundlagen* 4, 1958, 180-221.
- Heise, W., ALGOL—et Internationalt Sprog for Elektron Regnemaskiner, *Ingeniøren*, 68, årg. 17, 505 (1959).
- Bottenbruch, H., Erläuterung der algorithmischen Sprache ALGOL anhand einiger elementarer Programmierbeispiele, *Bl. Dtsch. Ges. Versicherungsmath.* 4, 1959, 199-208.
- Stephan, D., Die Algorithmische Sprache ALGOL 60, an Beispielen erläutert, *Bl. Dtsch. Ges. Versicherungsmath.* 5, 1960, 61-86.
- Bottenbruch, H., *A Critical Study of ALGOL*, Univ. Illinois, Digital Computer Laboratory Report 105, 60 Dec 8.
- Bauer, F. L., The Formula-controlled Logical Computer "Stanislaus", *Mathematics of Computation (MTAC)*, 14, 1960, 64-67.
- Dijkstra, E. W., Recursive Programming, *Numerische Mathematik* 2, 60 Oct, 312-318.
- Gibb, A., *ALGOL 60 Procedures for Range Arithmetic*, Stanford Univ., Applied Math. and Stat. Laboratories, Tech. Report 10, 61 Apr 12.
- Kudielka, V. et al., *Extension of the Algorithmic Language ALGOL*, 1961 July, Mailüfterl, Vienna, 34 pp. (U.S. Govt. Report DA-91-591-EUC 1430).
- Lucas, P., *Die Strukturanalyse von Formelübersetzern*, 1961, Mailüfterl, Vienna.
- Evshov, A. P., The Basic Principles of the Development of the Programming Program of the Institute of Mathematics of the U.S.S.R. A. S., *Siberian Mathematical Magazine* 2, No. 6, 1961.
- Bauer, F. L. and Samelson, K., Maschinelle Verarbeitung von Programmiersprachen (Processing of Programming Languages by Computer), in *Digitale Informationswandler*, Vieweg & Sohn, Braunschweig, 1962, 227-268.
- Nederkoorn, J., *A PERT Program in ALGOL 60*, Technical Report 56, Mathematical Centre, Amsterdam, 63 Feb, 22 pp.
- The Descriptor*, Burroughs B5000, Form 20002P.
- Extended ALGOL Reference Manual for the Burroughs B5000*, No. 5000-21012, Burroughs Corp., Detroit, 1963.
- SHARE ALGOL 60 Translator Manual*, No. 1426, 1577, SHARE Distribution Agency, IBM.
- ALGOL 60, An Introduction for FORTRAN Programmers*, Elliott Bros., London, 1963.
- Boothroyd, *A Guide to Machine-Independent Compiler Programming and ALGOL*, English Electric LEO, Kidsgrove, 1963.
- van der Mey, G., *Process for an ALGOL Translator*, Dr. Neher Laboratory, Leidschendam, Netherlands, Report 164 MA.
- Naur, P., (Ed.), *A Manual of GIER ALGOL*, Regnecentralen, Copenhagen, 1963.
- Ageev, M. I., *The Principles of the Algorithmic Language ALGOL 60*, Computing Centre AN U.S.S.R., Moscow, 1964, 116 pp. (revised edition, July 1965).
- Koster, C. H. A., *Efficient Rekenen in ALGOL*, Report of the Mathematical Centre, Amsterdam.
- Randell, B., Whetstone ALGOL Revisited, or Confession of a Compiler Writer, *Automatic Programming Information Bulletin* 21, 64 Jun, 1-10.
- Popov, V. N., Stepanov, A. G., Stisheva, A. G., Travnikova, N. A., A Programming Program, *Journal of Computational Math. and Mathematical Physics*, 4, 1, 1964.

- Shura-Bura, M. R. and Lubimskiy, E. Z., Translator: ALGOL 60, *Journal of Computational Math. and Mathematical Physics* 4, 1, 1964.
- Petrone, L., *Operators and Procedures in ALGOL-type Languages*, EURATOM, Ispra, Italy, EUR 2417.E, 65 May, 9 pp.
- Eyshov, A. P. (Ed.), *ALPHA Automatic Programming System*, USSR Academy of Sciences, Siberian Division, Novosibirsk, 1965, 264 pp.
- Cohen, J. and Nguyen-Huu-Dung, Definition de Procedures LISP en ALGOL; Exemple d'Utilisation, *Revue Française de traitement de l'Information* 8, 4, 1965, 271-293.
- Perlis, A. J., Formula Manipulation in Extended ALGOL, *Mededelingen v.h. Nederlands Rekenmachine Genootschap* 7, 6, 65 Dec.
- Thiessen, E., Automatic Conversion of BELL-programs to ALGOL-programs, in *Computing* 1, 4 (1966), 354-357.
- Iturriaga, R., Standish, T. A., Krutar, R. A., Earley, J. C., Formal Compiler Writing System FSL to Implement a Formula ALGOL Compiler, *Proceedings AFIP Spring Joint Computer Conference*, 1966, 241-252.
- Wirth, N., An Introduction to FORTRAN and ALGOL Programming, in *Mathematical Methods for Digital Computers*, Vol. 2, John Wiley & Sons, New York, 1967, 5-33.
- Bolliet, L., Auroux, A., Bellino, J., DIAMAG: A Multi-access System for Online ALGOL Programming, *Proceedings AFIP Spring Joint Computer Conference*, 1967, 547-552.
- Leroy, H., A Macro-generator for ALGOL, *Proceedings AFIP Spring Joint Computer Conference*, 1967, 663-669.

SUMMARY OF PUBLICATION OF FORMAL SPECIFICATIONS FOR THE ALGOL LANGUAGE

- Preliminary Report—International Algebraic Language:
in Communications ACM 1, 58 Dec, 8 (Note 1);
Annual Review in Automatic Programming 1, 1960, 268-289 (Note 1).
- Report on the Algorithmic Language ALGOL:
in Numerische Mathematik Bd. 1, 1959, 41-60 (Note 2).
- Report on the Algorithmic Language ALGOL 60:
in Communications ACM 3, 60 May, 299-314 (Note 3);
Numerische Mathematik 2, 1960, 106-136;
Annual Review in Automatic Programming 2, 1961, 351-390;
Acta Polytechnica Scand., Math. and Computing Machinery Series 5, AP 284;
Chiffres 3, 1960, 1-44 (French);
 Regnecentralen, Copenhagen, 1960, 40 pp.
- Revised Report on the Algorithmic Language ALGOL 60:
in Communications ACM 6, 63 Jan, 1-17;
The Computer Journal 5, 63 Apr, 349-367;
Annual Review in Automatic Programming 4, 1964, 217-258;
Numerische Mathematik 4, 1963, 420-453.
- Notes: (1) Equivalent to the CACM publication, not identical to the original ditto copy entitled "Zurich Conference on Algorithmic Language, Preliminary Report", 37 pp., due to further editing by Bemer.
 (2) Equivalent to the ditto report.
 (3) Reprints made available with typographical corrections as of 60 Jun 28 and 62 Apr 1.

SUMMARY OF ALGOL BULLETINS

Issue	Issue date	Pages	Issue	Issue date	Pages
1	59 Mar 16	6	14	62 Jan 16	18
2	59 May 5	8	15	62 Jun	
3	59 Jun 8	6	16	64 May	34
4	59 Aug 13	7	17	64 Jul	29
5	59 Sep 28	8	18	64 Oct	53
6	59 Oct 17	1	19	65 Jan	63
7	59 Nov 3	21	20	65 May	50
8	59 Dec 12	11	21	65 Nov	83
9	60 Mar 16	4	22	66 Feb	38
10	60 Oct 17	17	23	66 May	15
11	60 Dec 23	10	24	66 Sep	37
12	61 Apr 24	17	25†	67 Mar	30
13	61 Aug 18	13			

(Note: Starting with *ALGOL Bulletin* 16, produced under the sponsorship of IFIP WG 2.1, ALGOL, F. G. Duncan, Editor. Issues prior to this were produced by the Regnecentralen, Copenhagen, P. Naur, Editor.)

SUMMARY OF ALGOL BULLETIN SUPPLEMENTS

(Those published in regular journals and books will be indicated only by that reference)

Number	Mailed	Paper
1	59 Jun	Woodger, M., A Description of Basic ALGOL.
2	60 Mar 2	(The ALGOL 60 Report.)
3	60 Oct 20	Kerner, I., Bericht uber die algorithmische Sprache ALGOL 60.
4	60 Nov 30	(The Computer Journal 3, No. 2, 67.)
5	60 Nov 15	Jensen, J., Jensen, T., Mondrup, P., Naur, P., A Manual of the DASK ALGOL Language, Regnecentralen, Copenhagen.
6	61 Mar 17	(CACM 4, No. 1, 55.)
7	61 Mar 17	(CACM 4, No. 1, 59.)
8	61 Feb 3	(Computer Applications Symposium, 1960, 154.)
9	61 Apr 24	Naur, P., A Course of ALGOL 60 Programming, Regnecentralen, 38 pp.
10	61 Nov	Dijkstra, E. W., ALGOL 60 Translation.
11	61 Apr 24	(BIT 1, No. 1, 38.)
12	61 Jun 29	(The Computer Journal 4, No. 4, 292.)
13	61 Jun 26	(CACM 4, No. 1, 60, 65.)
14	61 Aug 29	(Input Language, see Book List.)
15	61 Aug 24	(BIT 1, No. 2, 89.)
16	61 Nov 6	Lucas, P., The Structure of Formula-Translators.
17	61 Oct 10	Youden, W. W., An Analysis of ALGOL 60 Syntax.
18	61 Oct 26	(Computer Applications Symposium, 1961, 115.)

† Also in SICPLAN Notices 2, No. 5, May 1967.

SUMMARY OF AUTOMATIC PROGRAMMING INFORMATION BULLETINS
(Automatic Programming Information Centre, Brighton, England)

Issue	Issue date	Pages	
1	60 Mar	2	
2	60 May	8	
3	60 Jun	8	
4	60 Sep	17	
5	60 Nov	11	
6	61 Feb	17	
7	61 May	36	(Special ALGOL 60 issue)
8	61 Jun	23	
9	61 Aug	12	
10	61 Oct	13	
11	61 Nov	30	
12	62 Jan	25	
13	62 Mar	15	
14	62 May	17	(Survey of Programming Languages)
15 } 16 }	62 Oct	31	
17	63 Apr	43	
18	63 Aug	24	
19	63 Dec	22	
20	64 Mar	4	
21	64 Jun	10	
22	64 Aug	15	
23	64 Oct	39	
24	65 Feb	14	
25	65 Mar	14	

(Discontinued 66 Feb, having achieved its aims. Richard Goodman, the Editor, died in August 1966 after a long illness.)

IFIP TECHNICAL COMMITTEE 2—PROGRAMMING LANGUAGES
(Chairman—H. Zemanek)

Meeting	Date	Locale
1	62 Mar 20	Munich
1	62 Mar 27	Rome
2	62 Aug 25	Munich
3	63 Sep 9	Oslo
4	64 May 11	Liblice
5	65 May 20	New York City
6	65 Nov 2	Nice
7	66 Apr 26	London
8	67 May 20	(Amsterdam)

IFIP WORKING GROUP 2.1—ALGOL
(Chairman—W. van der Poel)

Meeting	Date	Locale
1	62 Aug 28-30	Munich
2	63 Sep 10-13	Delft
3	64 Mar 16-20	Tutzing
4	64 Sep 14, 19	Baden
5	65 May 17-21	Princeton
6	65 Oct 25-29	St. Pierre de Chartreuse
7	66 Oct 3-8	Warsaw
8	67 May 16-20	Zandvoort

ISO/TC97/SUBCOMMITTEE 5—PROGRAMMING LANGUAGES
(Chairman—R. W. Bemer)

Meeting	Date	Locale	
1	61 May 18	Geneva	(as Working Group E)
2	62 May 9-10	Stockholm	”
3	62 Oct 9-13	Paris	”
4	63 Jun 5-7	Berlin	”
5	64 May 25-28	New York	(as Subcommittee 5)
6	65 Sep 6-10	Copenhagen	
7	67 Nov 6-10	Paris	

I am convinced that such transferability of programs, data, and programmers is within the present state-of-the-art. This panel from government and industry has been assembled to tell what has been done and what is planned.

Let us all cooperate to hasten the day when most programs written in any higher order language can be compiled and executed on most existing computers. The time, money and manpower saved by eliminating re-programming can then be used to solve other more interesting and useful problems.

Program transferability

by ROBERT W. BEMER

General Electric Company
Phoenix, Arizona

General

The problem of program transfer is such that most people think they understand the process better than they do. Optimism is rampant; success is elusive. I have some tenets which I believe to be *sine qua non*:

- Program transfer is complicated by each element which is different—user, CPU, configuration, operating system, etc.
- Programs must be planned for transfer. "After-the-fact" is virtually useless, like post-classification for information retrieval. The information loss is too high in the transfer from programmer to code. If everyone wrote and documented his program as a connectable black box, only the connecting process would need to be under the control of the user.
- In twelve years of hearing proponents discuss it, I have not yet seen successful mechanical translation of machine language programs. There are the processes which a translator:
 - a. Thinks it can do and can.
 - b. Thinks it can't do and says so, for human rework.
 - c. Thinks it *can* do and *can't*, and therefore *doesn't say so!*
- Transfer should always be made on a source program basis. Recompile is a trivial expense.
- To the highest possible degree, the documentation

of the program should be self-contained in the source program itself (rather than in the auxiliary documentation), and in a standard format and placement so that mechanized program tools know where to find the machine-readable information for extraction and use.

- Production of identical answers is (particularly for scientific problems) an additional requirement which must be specified and paid for. Differences may be due in part to differing internal arithmetic modes, but more often they are due to the overlooking of imprecision in method. On balance, obtaining different answers must be considered a healthy phenomenon.
- The criterion which a software module/component must meet in order to be self-documented adequately is:

"Can it be dropped into a program/data base for problem brokerage, whereupon a completely anonymous user may make a mechanical search to his requirements, find and use the module in his problem, and pay automatically a brokerage fee upon successful usage?"

This would be one standard that nobody would argue about—if he got found money at the end of the month, for conforming. Perhaps this might be a better solution than patenting software. Only thus can the non-specialist take advantage of computer utilities.

Some information required to transfer (run) a program¹

- Program name (number)
- Program function
- Descriptors, classification (computing reviews)
- Original computer system
- Original configuration, subset of required configuration, options used/available
- Other systems/configurations verified to run on
- Operating system, requirement, linkages, interfaces
- Running instructions
- Store requirements (resident program, nonresident program, data, tables, segmentation, overlay sequences)
- Source language (standard, dialect)
- Input/output data
- Data structures
- Data types
- Data elements, collating sequence

(1) To complete while producing the program.

- Interfaces (other units called, libraries)
 - Connections (via jumps, switches, natural flow)
 - Languages/processors equipped to call this program
- Method, average runtime (for interactive simulators)
 - Restrictions, constraints, degenerate cases, idiosyncrasies
 - Range, accuracy, precision
 - Changes occurring in conditions, status, original input
- Optional*
 - Information specific to program transfer
 - Default options—referring to international/national standards
 - Responsible organization
 - Grade of program (thoroughness of testing)
 - Test cases and answers (possible autoverification and answer match)
 - Bibliography, references
 - Copyright, price, etc.
 - Source/object program listing, number of instructions/statements

Mechanical tools for conversion²

- Combinatorial path exercisers through a program
- Programs which page the source code for the programmer and mechanically force him to be up-to-date
- Programs which mechanically check the linkage of units of a software system to provide a directed graph for flow verification, ensuring that any software unit will not interface with other software units to which it should *not* be connected.
- Mechanical determination of valid paths in the reverse direction of flow, as a diagnostic tool for finding "How did we get here from there?"
- Mechanical verification of successful meeting of interface requirements when passing from one software unit to another in a forward direction.
- Mechanical re-verification of linkage and interface requirements for any revisions.
- Code acceptance filters.
- A patch defense (correct/change in source code only)
- (De-)flowcharters

Mechanical capture of facilitating information³

The source-to-object program translation process

- (2) Used during the completion stage of the program, to prepare against transfer problems and to ensure a well-conditioned state.
- (3) To obtain in each use of the program.

yields information. Much of this is lost, but needn't be. Some of this information concerns elements which are not themselves standardized, but can be part of a standard list of measurements useful to program transfer.

Therefore a language processor should be constructed:

- To be self-descriptive of its characteristics (i.e., features contained, added or missing; dialects or differences).
- To affix to the original source program, as a certification of a kind, either an identification of, or its actual characteristics. It may also strike characteristics or features which were unnecessary for that source program.
- To inspect transferred programs for a match to its own characteristics.

If the transferred program is processed successfully:

- The identification of the new processor is also affixed to the source program.
- In any area where the new processor has lesser requirements (i.e., a smaller table worked successfully; a missing feature was not required), the affixed information is modified to show the lesser requirement.

Thus a source program, once processed, contains information on:

- The minimum known characteristics required for successful processing.
- All processors (with operating systems) which treat the source program successfully.

Software compatibility

by JOHN A. GOSDEN

The Mitre Corporation
McLean, Virginia

Data Exchange

There is a growing need for data exchange, particularly the passing of files of data between programs that were produced independently. This will be needed in the development of computer networks and data bases; for example, a head office installation collecting files from various divisions of a corporation to build a data base. Both the development of formal and informal computer networks as well as the economic feasibility of large

RB RECORD COPY

STRAIGHTENING OUT PROGRAMMING LANGUAGES

A PRESENTATION TO THE 10TH ANNIVERSARY MEETING OF CODASYL

1969 May 27 - 28

by

R. W. Bemer

The General Electric Company, Information Systems Group,
Phoenix, Arizona

THE REASONS

You are in a locked room with a bomb. It's a combination lock. There's a fellow on the outside who can tell you the complicated procedure necessary to open the lock and you can hear him. Unfortunately, he speaks only Italian and you cannot understand Italian.

B A N G ! ! !

That's the problem with languages. The knowledge of A is lost if B cannot understand. That's the problem with programming languages and really that is why CODASYL was started 10 years ago.

If we believe that the use of our intellectual resources is beneficial to man, then we must believe also that it is better to utilize those resources more efficiently. The writing of a program for a computer is an important example of the expenditure of intellectual resources.

Writing a single program does not cost so much; however, the magnitude is apparent when we consider our total inventory of programs.

TOTAL D.P. INVESTMENT - PLANET EARTH

Hardware - 1.0

Software - 1.5

Translating to money,

U.S. Installed Value - Hardware - \$16 billion

Non-U.S. Installed Value - Hardware - \$ 8 billion

\$24 billion

This implies \$36 billion in software!

I am not sure if the major reasons for developing programming languages were ever ranked. We know that we use COBOL because it is easier to write the program. We know that we use COBOL because it is easier for others to understand that program. We claim that we use COBOL because it helps us to transfer that intellectual resource to different equipment to perform the same function. If one would compare the inventory of COBOL programs (and I must confess I do not know what it is) with the entire 36 billion, he would see that we have in a measure failed.

We are approaching a new environment with these forcing functions:

- Separate software pricing permits mix and match of both hardware and software.
- Data bases enable information brokerage and load distribution.
- Transfer of software (representing large investments) to other equipment demands consistency of representation.
- Auxiliary use of computers at resource centers and networking, certainly for overload, and possibly to reduce local configurations to that required to run object programs only.
- Insulation of the user from hardware and operating systems.

John Haanstra has said that compatibility is not a goal, but rather a property which enables the result of data and program transferability. I have my own lemma that "if the data is not transferable, the program cannot be transferable".

It is quite evident now that the separate divisions of COBOL facilitate program transferability (or portability). However, we can and must do more for COBOL along this line and (more importantly) carry it to the other programming languages, both procedure and problem-oriented. The occasion of this 10th Anniversary of CODASYL, with its avowed intention of planning for the next decade, provides an appropriate forum for a proposal which could lessen the wastage for the next \$36 billion worth of software, which obviously will be produced over a shorter time scale than the first 36.

THE BACKGROUND

Presently there are two common types of programming languages--procedure-oriented (IFIP Definition J22) and problem-oriented (IFIP Definition J23). These definitions recognize an overlapping of terminology usage, which has been more complicated with the addition of languages for job control, data storage and retrieval, data communications, etc.

The structure of languages of the same class is also variable. FORTRAN does not have an explicit environment and data division; COBOL does. This matter is definitely overdue to be straightened out.

The key may be in the IFIP definitions for data and information:

A1 DATA A representation of facts or ideas in a formalised manner capable of being communicated or manipulated by some process.

Note: The representation may be more suitable either for human interpretation (e.g., printed text) or for interpretation by equipment (e.g., punched cards or electrical signals).

A2 DATA CARRIER A general term for a medium used to carry recorded data and intended to be easily transportable independently of the mechanism used in its interpretation.

Examples: A punched card, a magnetic tape, pre-printed stationery.

A3 INFORMATION In automatic data processing the meaning that a human assigns to data by means of the known conventions used in its representation.

Note: The term has a sense wider than that of ordinary information theory and nearer to that of common usage.

It is significant that these are the first definitions in the Vocabulary, but it was very difficult to reach agreement.

Let us concentrate on the distinction that information can be obtained only when one knows the conventions of data representation. This brings to mind a curious sequence of events--actually a cycle. The name CODASYL (the coinage of which was my small contribution) incorporates "data". When we started the standardizing bodies, we got a little fancier and said "Computers and Information Processing". With the marriage to communication and data bases, the plain facts are that we will process data and, incidentally, some information. Computer-based systems can move data around from place to place, put it away, find it again on the basis of its packaging, and (as in the case of cryptography, for example) perform transformations upon the data--all of these absolutely independent of the information content!

We won't have any analogy problems if we use the postal system as our example:

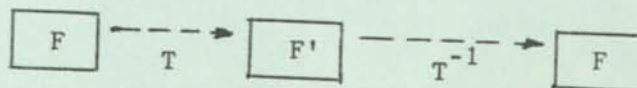
1. The mail carrier resides in an operating system environment--the Post Office system.

2. The carrier goes through a procedure, part of which is moving envelopes according to addressing on the outside.
3. He knows nothing about the information contained in the envelope that he is moving. Thus he cannot make procedural decisions based upon the information content. He can not peek into the envelope (ignore postcards, as he should).

We are now in danger of elaborating programming languages of the COBOL class for more data handling. By analogy we are giving the carrier some transparent envelopes and controlling his action through the content. It would have a terrible effect on postmen if they were required to decipher instructions in one manner when the contents of the letter are in English (read COBOL), and in another manner when the contents are in French (read FORTRAN), etc.

Another surprise to me is the appearance of separate proposals for a data manipulation language and a data communication language! Surely data movement is absolutely the primary enabling function in data processing. Why then are two separate languages required to cover the same function? I have tried to give a recapitulation of the fundamentals of data movement, trying in each dimension of description to give the universe of possibilities. It may not be foolproof, but so far it has fit every case tested:

1. Data Movement is accomplished by putting it in the form of a Message.
2. A Message is a bit string, with or without packaging. The packaging may precede or follow or both.
3. A message may be in original form F, or become F' via a known and understood transformation.



Some Examples:

- Digital-analog conversion (as for facsimile), the transformation being pulse or waveform to bit, and vice versa.
- Addition of parity
- Table lookup
- Scramble positions, or any encrypting
- Editing. The message XXXX may be formed from the original \$XX.XX.

4. A message may be interpreted or stated to be:
 - Data processed by the system, or
 - Instructions for system operation
5. A message may be moved:
 - Privately, in which case the packaging is not mandatory
 - Publicly, in which case packaging is mandatory
6. The information content may be known:
 - Privately, or
 - Publicly, via description in the packaging
 - Publicly, via standards of representation such as ISO R646 (USASCII), or registered alternates.
7. The information format may be known:
 - Privately, or
 - Publicly, via description in the packaging, or
 - Publicly, via standards existing and in derivation (e.g., magnetic tape labeling).
8. The message may be moved:
 - Physically, in space
 - Non-physically, in time (e.g., operative control transferred from one program to another)
9. The source may send either the original or a copy.
 The sink (destination) may accumulate the message or else destroy previous data to make space.
10. Any single data movement may have multiple sinks, but only one source.

Thus I contend that, although languages for job control, data storage and retrieval, data communication, and segmentation are all procedural, they must all have the property that they do not modify or lose the information carried in the data they manipulate. I would call such languages "Data Procedure Languages".

Remaining in the other class of procedure languages are COBOL, FORTRAN, ALGOL, IPL, and the like. These have sometimes been termed algorithmic languages. But, to highlight the present distinction, I would call them "Information Procedure Languages". I would go further and say that these should be limited to components which in fact operate upon data only with respect to the information content. As an example, the comparison statement:

```
IF CHARACTER EXCEEDS 'S' THEN NEXT STATEMENT OTHERWISE STOP.
```

Quite obviously (from the fact that NCR and IBM equipment operate differently for this statement) the information content is the relative position of 'S' in the alphabet, and not its data representation.

THE PROPOSAL

This separation of "Data Procedure Languages" from "Information Procedure Languages" is the motive power of my proposal. Data is our raw material. Software and hardware are only tools for manipulation. In some way the higher level languages (in the vacuum of not knowing enough about data structure) have achieved a disproportionate importance and a warped direction (one direction per language, in fact). Indeed, if I have a process to perform upon data, I may choose one of several information procedure languages. Conversely, more than one user of the same data should be allowed to operate upon that data by various information procedure languages.

Note that I say that this separation is the motive power. I didn't say it was a new idea. One of my old notes said "Check my old memos to support Grace Hopper on common data definition for all programming languages". Peter Landin's paper "The Next 700 Programming Languages" (66 March Communications of the ACM) concerned "A family of unimplemented computing languages...intended to span differences of application area by a unified framework". Professor Maurice Wilkes hit the problem again in his paper "The Outer and Inner Syntax of a Programming Language" (68 November issue of the Computer Journal) saying "There are two sides to a programming language; one is concerned with organizing the pattern of calculation, and the other with performing the actual operations needed". Unfortunately this did not get recognized by the reviewer as being very profound, for he said "The author seems to feel that this observation is justification for an article, and so continues for three pages with a quotation from Bertrand Russell, a fragment of the ALGOL 60 Report, and a humorous example intended to further belabor the point."

Well, today I have a wonderful chance to belabor the point again. I make the following 5-point proposal (not all of which depend upon the data/information separation):

1. Every program should depend, for its operation, upon having separate divisions for:
 - a. Identification
 - b. Environment
 - c. Data structure
 - d. Data procedure (not particular to the application)
 - e. Information procedure (specialized to the application)
2. For reasons of program transferability, economics, education, etc., all but the information procedure division should be common to all information procedure languages. (See VUEGRAPH) This whole framework gives what I call a "Composite Programming Language". This is the name of the recently created committee to which PL/I was assigned. If PL/I is a composite language, it should fit this pattern. That committee is welcome to take my paper here as a basic document.
3. The Environment Division should have provision for automatic affixing, after any compilation, of the imprimatur of that compiler, together with a statement/revision of the minimum actual requirements needed for the compilation of the program.
4. Every program should be permitted to contain more than one way of expressing the same function or action, only one of which will be compiled or executed conditionally. (See VUEGRAPH)
5. The five divisions should be transparent to (or inclusive of) mode of program operation such that:
 - A single switch setting will enable either reactive or batch processing.
 - A single switch setting will enable either checkout or run.

The purpose of the proposal is to have Programming Languages which can:

- Survive and exist in a larger world
- Permit program transferability
- Exist in a common structure and environment, to prevent ballooning of operating systems
- Adapt and assimilate new capabilities without impact or transplant shock (requires a sound structure for universality)
- Have features in common with each other, despite permitted dialectical differences

The proposal is not aimed primarily at compiler efficiency, but this may be a byproduct. Layering is usually a simple key which unlocks bigger problems. It reduces redundancy and permits arbitrary differences to atrophy. This is obvious from the work of Dijkstra, Gill and particularly Conway, who says the complexity of the system increases with the number of communication paths in the designing organization, which is combinatorial.

I do not mean to demand instant single standards. I favor coexistence to protect investment, but coexistence demands recognition! Recognition is not possible with implicit characteristics. They must be explicit. If something cannot be one way only, then the way must be identified. Some examples:

- Five different floating point precisions for System 360
- Duality required for phaseout of archaic or superseded features, such as the sign overpunch convention.

A switch can be set (or the environment division may signal the choice) for selective compilation. After sufficient atrophy the new version can be the default option.

CODASYL can do a maximum service if it takes as some of its goals for the next decade:

1. Further development of the data procedure languages now in process.
2. Addition to (and/or modification of) the environment division as may be required to accommodate the other information procedure languages. (1)
3. Addition to (and/or modification of) the data division to accomplish this same purpose.
4. Partitioning and reduction of COBOL so that only information processing features exist in the information procedure language, all others being reassigned to other divisions.
5. Ensuring that the bodies responsible for other major languages, and for new applications languages, make the modifications necessary to fit this framework.

This will yield a state where the elements of data procedure can be exercised by the information procedure only by a call and return, just like a subroutine. This leads to simplification possibilities in the operating system, which can take advantage of grouping of like calls. In other words

(1) Note: This usually includes implicitly the physical structure of the data in hardware, but possibly this could be taken out into its own division.

the Post Office sorts the mail and distributes it by route to the various postmen. When the data gets in your mailbox you may continue with your information procedure! In a multiprocessing environment this is more efficient than Special Delivery, exemplified by the READ verb in COBOL.

SPECIFICS

For a concrete example, let's look at segmentation and COBOL. COBOL was developed in a computing world which was essentially uniprogramming, and the specifications reflect this. At the NBS meeting of December 16, concerning a Federal COBOL Standard, GE took a firm position against having the present COBOL segmentation feature as either the primary or only segmentation feature allowed. Having had considerable success in multiprogramming and multiprocessing systems (as contrasted to nearly everyone else), we do keep our segmentation at the operating system level as a feature common to all programming languages and usage. For multiprogramming systems it is a general axiom that no specific feature of a programming language should override, usurp or endanger general features of the operating system to the degradation or failure of system performance.

The original concept of segmentation in COBOL was for overlaying in small stores. However, general segmentation is also used to:

- Split up compilation for efficiency (as the 600 does)
- Link subprograms compiled separately as written in the same or another programming language, having common access to common files (the CALL verb in FORTRAN, the LINKAGE-MODE in COBOL). This must be done at the level of the operating system (read Data Procedure), not the programming language itself. (read Information Procedure)
- Allow concurrent and interleaved running, in a multiprogramming/multiprocessing system, of completely independent programs compiled via different language processors. Here the segmentation is to allow the operating system to make decisions affecting the overall system efficiency and program mix.

Some of you will be familiar with the "run big" option of time-sharing systems. Keep in mind that he who binds segmentation at compile time cannot "run big" unless he recompiles for the configuration and resources available.

Aha!, you will say. Any smart compiler writer would hold this out from the COBOL compilation and assign it to the control of the operating system. But have they? If data procedures and information procedures are separated as I propose, they will have to.

DEGRADE PERFORMANCE

(USER PAYS! MOSTLY)

Another difficulty with segmentation is that it is a feature of Levels 3 and 4 only. Levels 1 and 2 were created specifically so that small computers could process COBOL source programs into object programs, yet when it comes to running these object programs it is these same small computers that need segmentation as much as, if not more than, the large computers processing Levels 3 and 4.

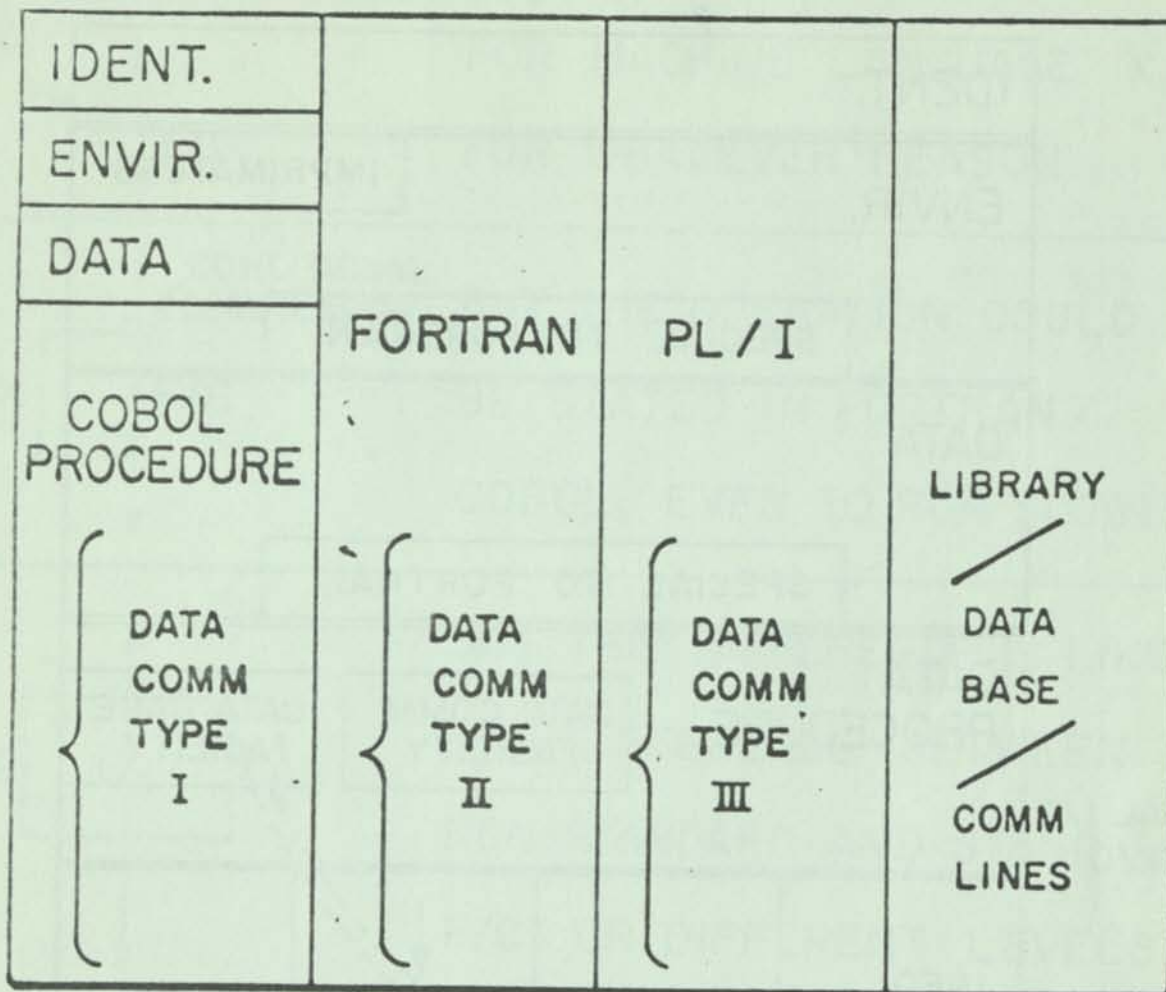
However, we think the real solution is not adding segmentation to Levels 1 and 2, but rather doing away with these levels altogether. Because most small computers are suitable terminals for large computers, there is no need to penalize the small computer by forcing it to compile for itself. It is suggested to write all source programs at Levels 3 and 4 and do remote compilation at what GE calls resource centers. Just because the COBOL source program is too big or too fancy for a small computer to compile does not mean that the object program is too big!

CONCLUSION

The concepts in this proposal may be simple, but I hold that they are profound. In one form or another they are certainly not original, but their time has come and my company has had much of the experience that proves them correct. Fortunately, the existing work of CODASYL would not be negated by accepting these concepts. Only a relatively small reorganization of specifications is necessary. However, a really big effort is necessary and unavoidable in order to bring all information procedure language into this common framework. I have intended to outline here a mechanism and plan for such a gradual, non-cataclysmic merging in a practical time frame, meanwhile inhibiting normal diversion.

I know of no group other than CODASYL that holds an official charter so unique to doing this work and I hope you will accept the challenge.

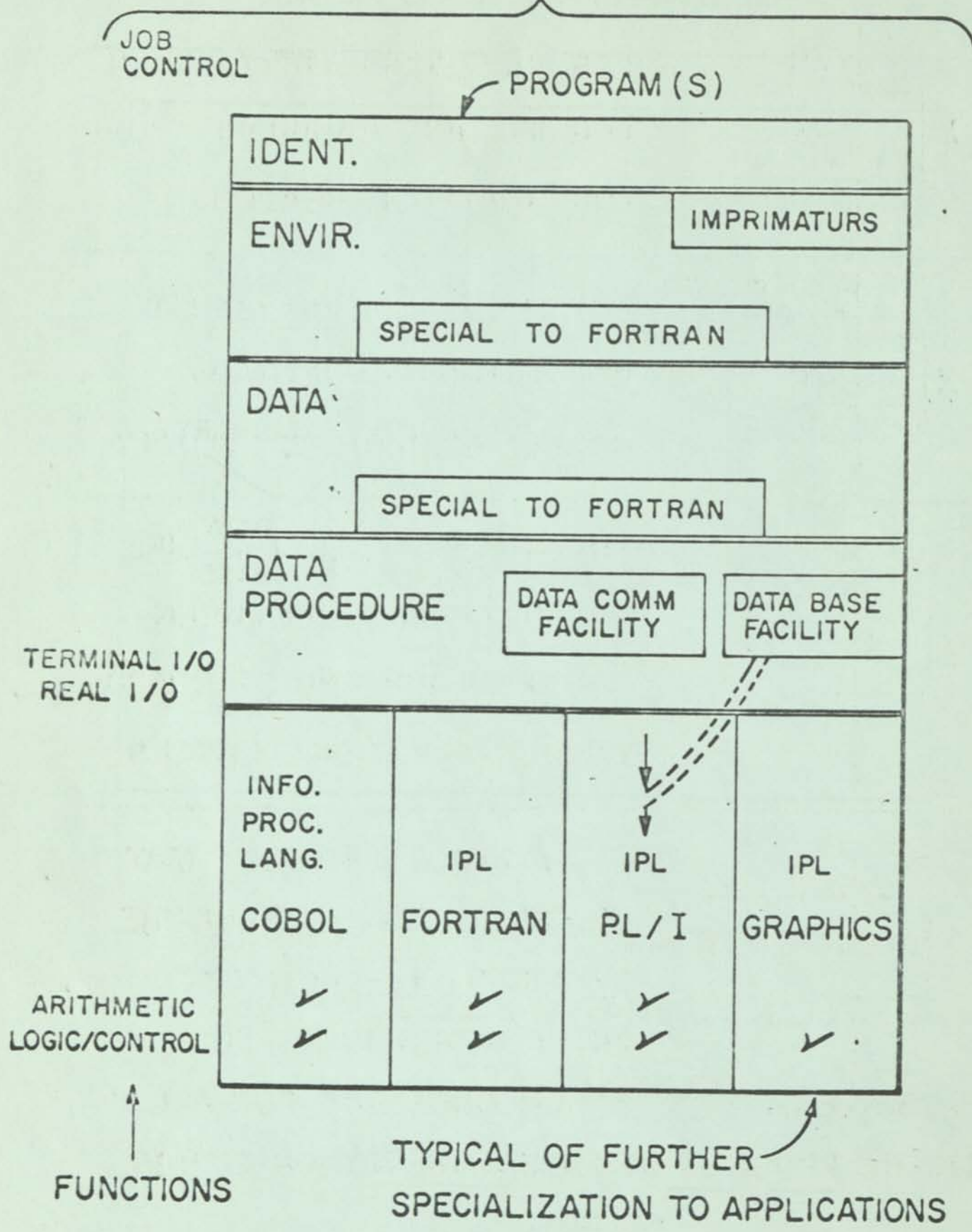
OPERATING SYSTEM



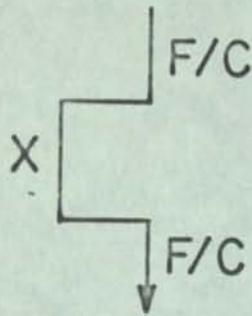
HOW TO PUT AN INTOLERABLE BURDEN
UPON AN OPERATING SYSTEM AND
DEGRADE PERFORMANCE !

(USER PAYS, MOSTLY)

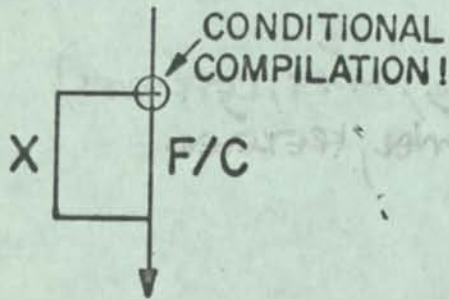
OPERATING SYSTEM



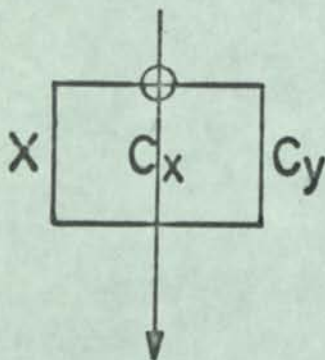
PRACTICAL COEXISTENCE



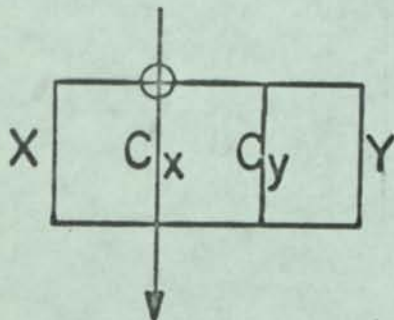
FORTRAN/COBOL PRECOMPILED
FOR MACHINE LANGUAGE X,
FOR WHATEVER REASON.



BUT THE OPERATION COULD
BE STATED IN FORTRAN /
COBOL, EVEN TO RUN SLOWLY



BUT THIS DIFFERENCE IS LIKE
THE DIFFERENCE BETWEEN
NON-STANDARD AND STANDARD
F/C, OR DIFFERENT LEVELS



IF C_y RUNS SLOWLY, Y MAY
BE CONSTRUCTED AT LEISURE.
MACHINES X & Y NOW ARE
FULLY REVERSIBLE, FOR:

- 1) CONVERSION IS SELDOM ONCE AND FOR ALL
- 2) THE PROGRAM MAY HAVE TO RUN ON EITHER
OR BOTH

DISTRIBUTION

- ✓ T. STEEL
- ✓ HEKIMI (THUS ECMA)
- ✓ ARNOT
- ✓ GOSDEN
- ✓ MOODERS
- ✓ INDUS. STDS SC (LAUR, DURAND, MARSHALL, SISCO, GREEN, BARTON)
- ✓ CONLES

76 ✓ P.T. WORKING LIST (S.B. WMS)

✓ ARR 7 RECIPIENTS NOT ON P.T. (EULSON, OLIVER, WILDE, LEFFOVITS)
MORNER, KREKELER

- ✓ LECHT
- ✓ BAYNARD (+ STEEL LTD)
- ✓ WEIL
- ✓ GROSCH, WHITE
- CLAMONS
- ✓ HUMPHREY, W.
- ✓ M. CLELAND, W.
- ✓ HAANSTRA
- ✓ L.P. ROBINSON
- ✓ BOB FOREST
- ✓ M. GREEN FIELD
- ✓ P.C. KLICK
- ✓ I. CLAUSSEN
- ✓ A. DEAN
- ✓ D. HAMILTON
- ✓ L. CEVENS
- ✓ S. ERDREICH
- ✓ ED. BERKLEY
- ANDRUS

the forum

The Forum is offered for readers who want to express their opinion on any aspect of information processing. Your contributions are invited.

ESCAPE TO REALITY

Along with nearly everyone I approve the advantages to be realized from the further integration of computers and communications. Some speak of it as a marriage; if so, it will take more planning than we have done so far to keep this marriage from floundering! Observe the following:

••

Know what it is? It's a cross section of a twisted pair of wires, a very basic element of communication systems. Seen from this aspect, the data communication system does not know who or what is on the receiving or sending end unless the establishment of the hookup, or linkage, defines these explicitly. This is why such extensive work has been undertaken in the standardization of data communication control procedures, facilitating movement of data from one place to another. However, we may still be blind to the format and meaning of the data. This is why much work is overdue on data descriptive languages. Let us consider the time when we have full standards for both communication procedures and data description. Are we then in full control? Absolutely not! The reason is that we have an inventory of almost \$40 billion in mechanically recorded data and software, and well over 99% of it is not accompanied by any explicit description of what it is—either in encoding or format. Data communication, as we speak of it now, occurs via public utilities. Therefore, using computers adjointly requires that data and programs have public (explicit) identification (this does not exclude reverting to a private mode later, just as personal defenses may be dropped once we establish that we are

communicating with a friendly party). Obviously we won't wish to keep the full inventory of software, but what should be done to salvage some¹ of the rest?

Anyone who thinks that we can move overnight to these new standards (when they arrive) is out of touch with reality.



There is information loss from problem analysis to programming, from source to object program, from a complete form of the data to the keypunching form on the card (e.g., \$23.57 to 2357), and from the logical file structure to the physical file structure determined implicitly by the equipment and the program. Miracles are still difficult to find, but there is a workable mechanism that

offers graceful coexistence and eventual conversion to a unified system. It is called "ESCAPE."

ESCAPE (abbreviated ESC) is a character which should be universal (although it may be difficult to add to certain 6-bit codes). In the 7-bit codes of ISO Recommendation 646, and ASCII, it is represented by: 001 1011. When ESC is encountered, the normal (implicit) meaning of the following data stream is disabled. However, the following characters have R646 meaning until an "ESCAPE sequence" is completed. This sequence consists of ESC, a number of intermediate characters (I) and a final character (F). The characters (I) and (F) are selected from the ISO code and are mutually exclusive sets. The best description of this mechanism is found in ECMA (European Computer Manufacturers Association) Document TC1/69/14.

Most present usage for ESCAPE sequences is for changing the meaning of the coded character set following. Thus: ESC & (F) indicates that the message which follows is not in ISO (ASCII) code but rather in EIA code RS 244 for numerical control of machine tools. EBCDIC, packed numeric, floating point for binary, etc., should all have their particular ESCAPE sequence.

Perhaps a more important usage is the indication of data formats (eight intermediate characters remain unassigned now for 3-character sequences).

As an example, I have proposed that every magnetic tape label begin with: ESC / 1. This says to read the label according to the first USA Standard for Magnetic Tape Labels. If some day we should wish to amend the standard for such labels, how would the software accept data from tapes labeled in the original way and also tapes labeled the new way? Simple, for the new label form begins with: ESC / 2.

Suppose that Social Security permits updating of filings via communications systems. Disc packs have formats different from magnetic tapes, yet looking at that twisted pair

••

how would the Social Security computer know how to accept the data, unless perhaps disc packs had the ESCAPE sequence ESC / 5?

Here ESC/ triggers a programmed

¹ Obviously if program and data are inextricable, as in the formatted read/write commands of FORTRAN, this may be very difficult. In many cases special conversion runs may be necessary.

You solve our problems, we'll solve yours.

If you're the kind of data processing professional who can apply imagination and ingenuity to challenging problems, consider some of the advantages of a career with McDonnell Douglas.

We have McDonnell Automation Company DATADROMES™ in St. Louis, Denver, Houston, Los Angeles, New York, and Washington, D.C. They offer opportunities in commercial data service, engineering, scientific and business computing and data processing.

And we have similar openings at our Douglas Aircraft Company or with McDonnell Douglas Astronautics Company's Information Systems Subdivision in Southern California.

We're looking for scientific programmers, math modelers, digital computer analysts, business programmers, consultants, systems analysts, marketing representatives and sales engineers.

If multi-project problem-solving is the kind of challenge you're looking for, just send the coupon, with your resume if available. We will arrange an interview.

Douglas Aircraft Company, Mr. P. T. Williamson, Professional Employment, 3855 Lakewood Boulevard, Long Beach, Calif. 90801.

McDonnell Automation Company, Mr. W. R. Wardle, Professional Employment, Box 14308, St. Louis, Missouri 63178.

McDonnell Douglas Astronautics Company, Mr. N. T. Stocks, Professional Employment, 5301 Bolsa Avenue, Huntington Beach, Calif. 92646.

ESCAPE TO REALITY . . .

table lookup to see what subroutine should be used to read the particular format identified.

Coexistence demands recognition, yet I have stated that present data and programs are not recognizable out of context. Suppose that the computing world were to follow these steps:

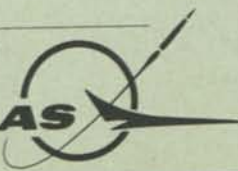
1. ESCape sequences are proposed and registered for the various data and program forms that are fairly common (less common forms need only the private and unregistered sequences, indicated by two or more intermediate characters).
2. Software is modified to detect omission of these sequences, labels, etc. The original or other programmer is requested to supply the missing information.
3. Software replaces the implicit form of the data or source program by the explicit form. Mixed alphanumeric and packed numeric data are physically separated by ESCape sequences, where formerly only the program knew where the split was. Probably no use is made of such identification at this time, and actual control may be bypassed.
4. This process occurs for a number of years, and more and more data and programs are converted to explicit identification. New data and programs are created similarly. After some years the conventions and standards are really established.
5. New software systems are designed to operate with identification control. Data and software are recognized either as standard, or as non-standard of a certain, identified type. If nonstandard, the system determines:
 - a. That it can handle the representation and format, and possibly convert it to the standard in the process, or
 - b. That it cannot handle it, and must call for help.

I hope the industry agrees with me that this is a practical approach and should be undertaken. It doesn't work 100%, but then it works much more than 0%. Don't forget, the communications industry found out long before we entered the act that having self-identified devices makes things a lot simpler!

—R. W. BEMER

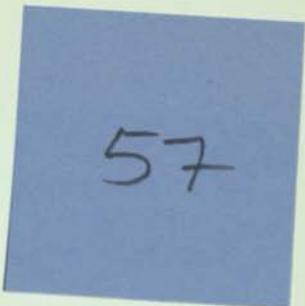
IE TET
COM
OCYOWT W
I T TOO YD
{OCR}

Name _____
Home address _____
City & State _____ Zip Code _____ Phone _____
Education: BS _____ MS _____ PhD _____ Major Field _____
(date) (date) (date)
Primary experience area _____
Present position _____
Area choice: East Midwest West Best Opportunity

MCDONNELL DOUGLAS 
An equal opportunity employer

CIRCLE 332 ON READER CARD

7571



Software Engineering

COINS III

VOLUME I

*Proceedings of the Third Symposium on Computer and
Information Sciences held in Miami Beach, Florida, December, 1969*

Edited by

JULIUS T. TOU

Center for Informatics Research
University of Florida
Gainesville, Florida

510.7834
T642.A



Academic Press New York · London · 1970

Manageable Software Engineering

R. W. Bemer

GENERAL ELECTRIC COMPANY
PHOENIX, ARIZONA

I. Introduction

Management problems exist for software systems, not for small components and subroutines. There is little difficulty in designing paper systems, but a system that is to be built and used demands extensive management. Thus the possibility of managing successfully a large software project becomes a competing design criterion. The constrictions of communication, control, decision, and tradeoff increase nonlinearly with project size, often becoming of such magnitude that they outweigh technical design choices that are apparently independent.

It is evident that this has not been recognized adequately in the design and construction of many previous large software systems. There is no need to name these failures here; many of us would like to be spared the pain of memory. It can't be size alone that causes these administrative difficulties, for there are many examples of successful administration of large projects. Is there something special about software that introduces new management traps, such as: Invisibility? Intangibility and poor definition of the task to be done? Micro time scale of component actions with respect to time scale of human interaction, thus masking inefficiencies? Entwinement of the engineering, manufacturing and distribution functions? Inability to prove correctness? But these are true to varying degrees in other fields. To my mind, they represent only excuses for the real problem, which is: As a relatively new profession (?) we are obsessed with reinvention, and forget that there is something known as management science. We fail to go through a simple exercise that should be standard for all development, to ask a series of questions that are not answered fully until there is feedback from the next question. The newspaper business has its formula of Who, What, When, Where, How. Similarly, the software producer should answer the questions of Table I. One may quibble with the percentages given, but these questions form the structure of this chapter, and are treated in turn in the next seven sections.

TABLE I

Question	Decision completeness, %
1. What should be produced?	5-10
2. Should it be produced?	30
3. Can it be produced?	70
4. How should the producer be organized?	80
5. How should the product be tested?	90
6. How should the product be introduced?	95
7. How should the product be improved and serviced?	100

II. What Should Be Produced?

1. *Does it Fill a Need?*

Answer this question carefully. At a 1958 GUIDE meeting, it was reported that a user programmer had rewritten an IBM input routine to run 10% faster. Based upon programmer cost, machine time for test, and percent of usage, it would pay off in the year 2040, at which time not many 705's will be around. A trivial case, perhaps, but the McKinsey report [1] shows that this applies also on a larger scale. Do not be afraid to discuss DP systems with your management. They have found out how much they cost, and will probably listen carefully.

Do not be too ambitious initially. Goals can change as you go along, and there is nothing with lower salvage value than a DP system which does a job you do not want, and is too difficult to modify.

2. *For What Market?*

Is it for use within your own company, or can some generalization or modification in design enable it to be sold to others for the same purpose? Or can the algorithm be compartmentalized from the application so it may be used for different purposes by you and others [2]?

3. *What Are the Advantages and Disadvantages, such as Efficiency and Cost Effectiveness?*

Beware the apparently aesthetic choices; don't forget production costs, use costs, life cycle, durability, reliability, and maintainability. Make sure that the tradeoffs are expressed quantitatively.

4. *What Are the Characteristics and Side Effects?*

Because software is supposed to educate the computer to do useful work in conjunction with humans, it should fit human capacity and characteristics. Some notable failures have been caused by ignoring this requirement.

The software system almost always overshadows the hardware system, and should be treated accordingly. Raw hardware power can be degraded as much as 90% by improper software.

III. *Should It Be Produced?*

1. *Will It Pay for Itself?*

Adjust the projected gain—for optimism, costs of conversion, introduction, and disruption of continuing processes.

2. *Will It Be Useful When Introduced?*

Large software systems have long production cycles, which are commonly underestimated. David [3, p. 69] says, "In the past (and probably in the foreseeable future) estimates of the effort (man-years) to complete tasks involving new software concepts are likely to be low by factors of 2.5 to 4." Periodic reviews are useful during the production cycle to see if the original assumptions still hold. An added margin of flexibility at design time pays off in enabling adaptability to the dynamics of change.

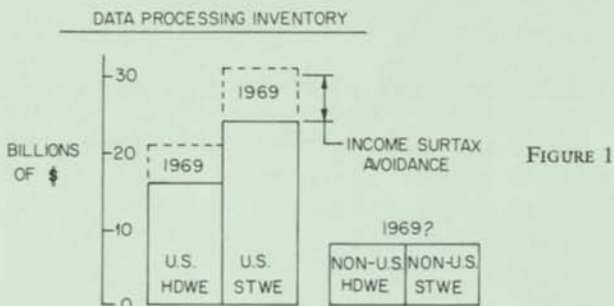
3. *Is It Timely?*

There is danger of missing the right point in technology, either too soon or too late. One should question if it is possible to get along without it until a jump to a new technique can be made. This is a question of best strategy, and all factors of the business should be considered.

4. *Make or Buy?*

It may be possible to get it elsewhere, in whole or in part, either cheaper or at the same cost. Figure 1 indicates that there is a vast amount available, although much of it is not portable to other equipment or installations.

Salable software packages are in the ascendancy, although most still have portability difficulties. Question #1 of this section may have a different answer if the software is planned for resale. The best sources for obtaining



outside software are user associations, software houses, and trade associations.

Remember that usually only one of a kind is necessary. Don't buy, externally or internally, more than is necessary. I know one operating system that has 20 different GET/PUT routines in it, by 20 different programmers.

5. How to Make the Final Decision?

I have not seen a quantitative answer to this question. When it was first asked of me, at IBM in 1957, the reply was to get a man with the best batting average in extrapolation, and trust proportionately to his judgment. This still seems the best answer to such a complex question.

IV. Can It Be Produced?

1. Is It Possible at All?

My most lasting impression of J. Paul Getty came from a Playboy article in which he said that the smart man does not take on the impossible. There are software systems which are neither feasible nor possible to build, given even unlimited resources of programmers and computers. There are methods at two extremes: (1) Plan the system all at once, then build it all at once. (2) Follow my five word motto, "Do something small, useful, now", with of course an eye to the changing future.

I don't have much faith in the first method, particularly for data processing, because every big management scheme I have seen has died for two reasons: (1) The planning was so monolithic that it took so long to do that it was out of date before it could be implemented. (2) Even then it could not be corrected or modified because the lack of results led management to put the planners out of a job. Obviously, the function was being performed somehow during this period.

Assuming all else is OK, one should keep an eye upon the permanence of his management and its goals. A new boss will often redirect effort and restart from nearly scratch. This is common for elected public officials.

2. *Are the Resources Adequate?*

Here we speak of all resources—money, talent (not manpower), time, technology, and direction—and they must be all allocatable to the project. Be careful when offered miracles. For years, I have carried a little cartoon in my billfold. It shows two programmers looking at a printout, and one says "Hey, Joe! It says our jobs are next!" Don't you believe it; we couldn't get rid of those two in any way!

Figure 2 is the McClure chart, from the Report on the NATO Software Engineering Conference. It shows how many instructions you will get if you ask a manufacturer to give you his software for a certain system. Note that the vertical scale is logarithmic. This makes me fear that in a few years not only will Joe and his buddy still be around, but so will twenty more programmers, and there may not be that much suitable programming talent lying around loose, even with a massive educational effort.

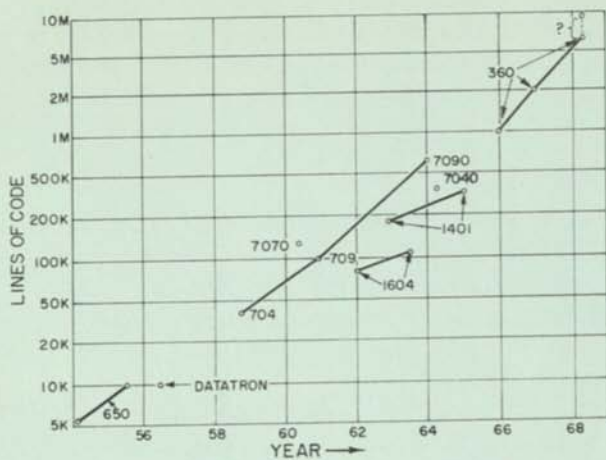
Is the size of a software system a worrisome factor? It certainly is, because productivity for basic software has not increased with system size. If anything, it has decreased markedly, and this is not surprising when one considers the inevitable increase in connectivity [4]. Figure 3 is my compilation of some productivity statistics, in terms of instructions per hour (both scales are logarithmic). This chart is designed to reflect total budget figures on the basis of approximately 30% for design and implementation, 20% for test, and 50% for management, documentation, and support. I have arrived at the OS 360 figure in several consistent ways, which are worth enumerating here:

1. Conway [5] postulated an expenditure of 15 million dollars in 1963, 45 million dollars in 1964, and 60 million dollars for the years 1965–1968. This is consistent with official IBM figures as reported in *Fortune* magazine for 1966 October. This yields 300 million dollars to produce the 5 million instructions the McClure chart shows for the end of 1968, or 60 dollars per instruction.

2. Original information released on the 360 software was 160 million dollars for about 3 million instructions, as produced by 3000 programmers at peak. This averages 53 dollars per instruction.

3. An IBM spokesman asked me at the 1967 February SHARE meeting, "Would you believe \$53.50?"

4. Assume programmers at 20,000 dollars per year for 2000 hours of work. At this 10 dollars per hour rate, 0.2 instructions per hour would cost 50 dollars, which is quite consistent.



GROWTH IN SOFTWARE REQUIREMENTS

FIGURE 2

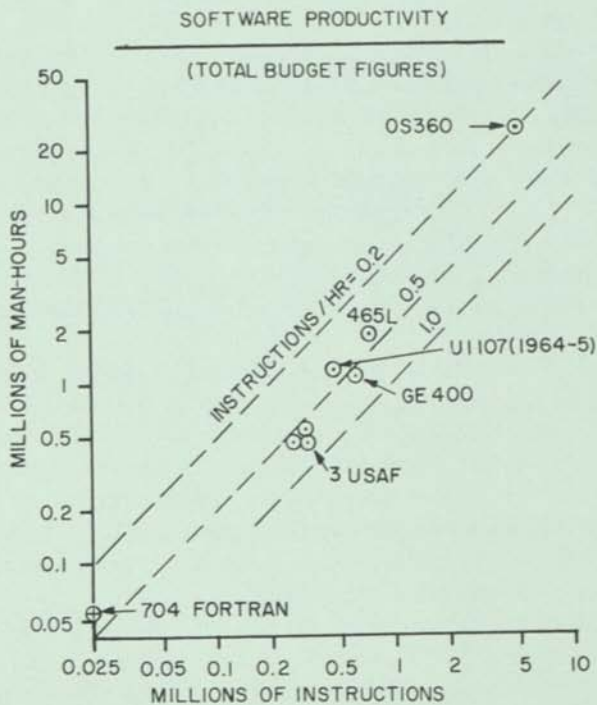


FIGURE 3

5. 3000 programmers at 20,000 dollars per year yield the 60 million dollars per year figure that Mr. T. J. Watson gave to the 1966 March meeting of SHARE.

These production figures will seem low to many. One should not forget that they are for very large, mature systems of basic software [6]. The cost of an instruction rises with longevity, because these systems must be maintained and enhanced. Some parts are rewritten several times, and the superseded instructions can't be counted anymore, even though their production cost is still a factor.

My nightmares come from imagining a new system scheduled for 1972. If the McClure chart holds true to give 25 million instructions, then the best figures we have say that it will cost a billion and quarter dollars, produced by 15,000 programmers.

An obvious point of rebellion is "Are all those instructions necessary?" Wouldn't it be nice to find some deadwood? According to David [3, p. 56], this is very possible, as demonstrated in the MULTICS system (Table II).

TABLE II

WHY SO MANY INSTRUCTIONS? MULTICS—1,000,000 Reduced to 300,000

Module	Improvement		Man/mo. effort
	Size	Perf.	
Page fault mechanism	1/26	50/1	3
Interprocess communication	1/20	40/1	2
Segment management	1/10	20/1	.5
Editor	1/16	25/1	.5
I/O	1/4	8/1	3

One would hardly express it as a law, on the basis of so few samples, but in these cases it appears that if the program is $1/N$ its former size, it will run $2N$ times faster. Part of the excess was due to use of a higher-level language, of course, but this should not be used to discriminate against higher-level languages per se. The sin is in using them in disregard of hardware characteristics. Code expansion is not the only culprit; duplication and unuse constitute an area of very high potential for extermination of excess instructions.

Now, if we can get a defined minimum of useful instructions to produce, let's consider the people that are going to produce them. Figure 4 is a serendipity product of Sackman *et al.* [7]. In explanation of the serendipity, these results came from an experiment to measure the effects of on line programming versus off line, but differences between individual programmers were

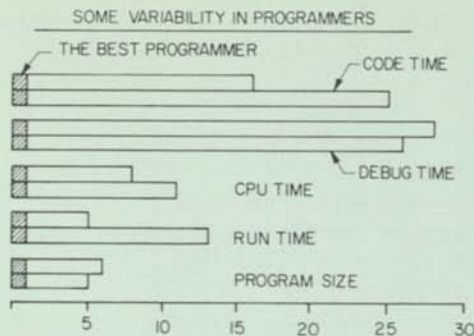


FIGURE 4

so great that they voided any possibility of measurements for the original purpose. Two identical problems were given to a group of twelve programmers with an average experience of seven years.

I told the authors that I considered the paper in which these data appeared to be the most important work in the computer field in 1968; this opinion is unchanged. Here we see more justification for asking if all those instructions are necessary. Naturally, not all the worst cases in each category are due to the same bad programmer, but the correlations are somewhat monotonic.

Note that the product of CPU time and program size would be the degradation factor in a multiprogramming system. This speaks strongly to the position that the best programmers should be selected and screened for the production of basic software. So does the following consideration, which treats not the originating cost, but the cost of use by the computer world:

During early instrumentation, 7% of GE 600 FORTRAN compile time was found to be in four instructions, easily reduced to two (3.5% saving). Suppose this were true for all software on third-generation systems, with an installed value of 14 billion dollars. Now, if 10% of use is FORTRAN, and 40% of that is compilation, then two redundant instructions waste

$$14 \text{ billion dollars} \times .1 \times .4 \times .035 = \$19,600,000$$

or about 10 million dollars per instruction!

To me, this is a frightening picture, for these are unrecoverable costs to the user. This should not be passed off as an isolated case, for it is demonstrable that most software systems and application programs are honeycombed with waste elements that surpass this one. And don't think that General Electric is embarrassed to disclose this case—after all, we have taken *ours* out by intensive application of instrumentation programs! We're even proud, considering that congressmen have been reelected with ease for saving the public this much money! In fact, there are potential savings of over a billion dollars to be realized by demanding instrumentation and measurement of software.

Thus, we see one critical input to the pragmatic question, "Can it be produced?" Many people have given recipes for extrication from this dilemma, all put forward with great fervor and, inversely, little hard justification. My list is:

1. High-level languages to write in (no one will quote more than a 3:1 advantage, and we have seen how that can be abused).
2. Good software management. (If you can find it, train it, have time to train it, keep it, and keep it programming!)
3. A software production environment (the factory).
4. Good programmers.

Some may think I have listed these in decreasing order of importance. To the contrary, it is in *increasing* order. The good programmer is the key, just as the top engineer is the key for hardware. The Univac 1004, a very successful piece of equipment, was designed and built by not more than a dozen people in what was called "The Barn", in Rowayton, Connecticut. As an aside, they did not build it to fit the existing market; their product shaped the market!

3. *Are the Production Methods Available?*

The generality of a data-processing system makes possible the finest production methods, yet these are seldom exploited to advantage. A major drawback of large software systems is that a substantial portion of the production cycle is often wasted by the invisibility of software—when the programmer finally builds something, we find it is not what we wanted. Then we must build something else, and the lost time cannot be regained. Large systems are too complicated to depend upon intuitive design, or for one individual to comprehend totally, or for a group of people to cooperate in the construction of and communication about without mechanical aids. An ideal plan is to build a model or skeleton and, if it acts as we wish, then to replace the simulated units by real units—carefully, one at a time, to avoid confusion.

Production identification, change control, and labor distribution are important tools. Their very tangibility for estimating provides the capability of recalibrating to better estimates. They also make it more difficult for programmers to lie to themselves, and perhaps to their management. I also have a personal predilection for standing in a machine room and sensing what is really happening. Then this can be matched against the production control, scheduling, and costing. PERT has failed in many large software projects, sometimes because there was not enough time to provide the inputs, sometimes because it only gives the latest time one can do something provided absolutely nothing goes wrong with all of the other things that people waited until the deadline to do!

4. *Is a Good Production Environment Available?*

If computers are useful for a general class of problem, then they should be useful for producing their own systems. Some manufacturers are now building a "software factory", or an environment residing upon a computer within which all software production takes place. Programmers are directly on line via terminals, and keypunchers are bypassed.

It is true that such systems will be very useful eventually for management control in large projects, but in the beginning the accent and priority for delivery must be on service to the programmer. The programmer is independent, so we must make the system attractive to him and worth the usage. One of the most important aspects is increasing the number of accesses to the computer per day. This is as low as one or two in much of our industry. At this slow pace, programmers tend to lose the thread of their thought and spin wheels. The difference between the good and bad programmer may well rest upon the need for cohesiveness and pattern. Perhaps it is like half-life decay of radioactive materials. In any event, this is a crucial factor in success of large projects—yet it has been treated as virtually unimportant!

Greg Williams of GE has proposed a lesson for management on why software is so expensive under limited-access conditions. He would ask them to use the BASIC time sharing system to convert clock time to Gregorian, to head an output, or to an accounting system. A simple, everyday problem, but he hasn't had the nerve to try it yet. His estimate for management—100 mistakes, finding two per day. That is a long clock time!

V. *How Should the Producer Be Organized?*

There is no one answer to this question, and perhaps more than one hundred. Software mirrors intellectual processes, which are capable of infinite variation. Some organization is required, for large software projects must be subdivided. It is important to observe Conway's law [8], that the form and size of the product reflects the form and size of the planning organization. The sub-organizations should be structured to follow the design and architecture, with the program interfaces under control of higher management.

1. *How Large Should Modules Be?*

Several authorities fix this figure at from 400 to 1000 instructions per programmer. If this seems low, recall that a gross of 0.2 instructions per hour means about one per hour by the programmer actually writing them, so that 1000 instructions represent about a half-year's work. This figure may be raised significantly for smaller projects with fewer interactions, but seems to hold well for systems of more than 250,000 instructions.

2. *Who Should Be in Charge?*

Software engineering has much to learn from hospitals, where the doctor does the work, with his decisions generally overriding the administrator's. For large projects, I favor a leader who is a working programmer, not just a supervisor. If possible, it should be at least his third project of that type, although he need not necessarily have been in charge previously. The reason for this is that the first time he reinvents, ignoring literature, competition, and scrounging; the second time, he is too confident that he can avoid all the mistakes made the first time. The GECOS III operating system for the 600 is a splendid example of such avoidance of the Peter Principle, which takes us to another question:

3. *How Should Design and Implementation Be Partitioned?*

Here we run head on into the old argument about system analysts versus programmers versus coders. Contrary to intuition, such a division may make sense for small projects, but not for large systems! A much more careful and practical design will originate from the man that knows he will be stuck for a year or more in its production! Additionally, the reasons for the design are so much in his cognizance that he is alert to signals that a design change may be desirable.

4. *How Does the New Project Coexist with Present Work?*

Present stratification is usually by job title or project assignment. Perhaps we should try to stratify the individual programmer, giving him concurrent responsibilities in several aspects—design, implementation, maintenance of his previous work (at least on call), and, in the case of software houses, assistance in customer sites. Admittedly, certain projects may be too complex to permit distraction, but these are few. A side benefit may be found in closer connection between present and future software, in the area of data and program transferability.

VI. *How Should the Product Be Tested?*

The proper design and use of extensive testing is mandatory. Auto manufacturers have their test tracks to detect failure and weakness before they make multiple copies to be driven by customers over whom they have no control. In fact, the French do call software testing "rodage". This testing becomes even more vital with separate software pricing, and also with con-

siderations of public welfare and safety as computers become further integrated into human activity. This means that a nontrivial portion of the total production costs must be allocated to this function.

1. *What Should Be Tested?*

The two major categories of quality standards are performance and compliance. Unfortunately, the first is only now getting its full share of attention.

2. *What Are the Testing Tools?*

A few tools are enumerated here; many more are possible and in use. Performance testing is necessary because any given process can be: (1) unnecessary, (2) done more times than necessary (i.e., rerun), (3) too slow due to hardware, (4) too slow due to software, (5) too slow due to hardware-software imbalance, (6) undesirable, but imposed by conflicting or non-existent standards, (7) inutile because of logic conflicts, (8) satisfactory.

Any of these can occur because of: (1) basic system software (the supplier should fix it); (2) application usage (the supplier should advise and also control the default options for preferred usage when possible).

For performance testing, the tools are:

1. Standards of comparison. With parameters of hardware performance (such as Gibson mix), number of object instructions, and precision of input and output, certain common functions can be compared against what is considered good quality in the industry. For example, if the sine-cosine evaluation routine runs extraordinarily slow compared to what comparison standards say it should do, it should be considered for rework.

2. Periodic instrumentation, either by hardware (zero time) or software (finite time, not supportable continuously). Hardware instrumentation is accomplished normally by tapping in a second computer system or a special hardware device. Software instrumentation can consist of: (a) interface tracers, for connectivity; (b) trapping analysis of module use, timing, control acquisition and release, etc. [9-12], (c) hardware-initiated actions for later software analysis; as an example, Ellison set the rundown timer on the GE-600 extraordinarily low, so that a given process could barely get started before interrupt and relinquishment of control occurred; in this way, a normal 24-hour use took almost 48 hours, while the actual store location of the instruction being executed at interrupt time was recorded. A later count and distribution gave an excellent Monte Carlo simulation of the frequency of use of the various software modules. (This was how the anomaly reported in question 2 of Section IV was detected.)

3. Continuous instrumentation (low and supportable time allocated to this purpose). This would include: (a) gathering statistics during operating system time for later analysis; (b) monitoring resource allocation and usage for real time display to the operator, preferably by CRT.

For compliance testing, the tools are:

1. Generalized tests for well-known standards, such as the U.S. Navy COBOL Certifier.

2. Special tests written (concurrently with the software production cycle) to test conformity to specifications. At Bull General Electric, I had at least one programmer in every ten allocated to such tasks; not a surprising ratio considering that the testing function for large systems can use as much as 20% of the total budget.

3. Test cases. Formerly, these have been considered as primarily for application programs, but they are also particularly valuable for testing successive system revisions. One accumulates a test file of the malfunctions reported for previous versions of the system, together with a sampling of small applications. The file resulting from processing with the new system is mechanically compared with the previous answer file, and deviations displayed for analysis.

Quality in both performance and compliance is checked by field test. One would wish to avoid such a procedure if possible, but most of the time this is impossible for large systems, time sharing systems being a particularly visible example. It is not known, nor have the computer scientists provided us with the insight, how to simulate and test a large multiaccess system by means of another computer program that exhibits the real time properties of: (1) any randomly possible selection from the U.S. communication system, (2) the U.S. population making other demands upon that system, (3) an unpredictable user population, either in loading or arbitrary usage.

My company has found that many strenuous measures must be taken to check out new time sharing systems. We have even switched a large number of internal users from the regular national system to the field-test system, to provide instantaneous overloads, peculiar usage combinations, and time-of-day variations.

3. *When Should It Be Tested?*

Quality Control is continuous testing, during production, by the producer. Quality Assurance is discrete testing, after production, by an agency acting on the behalf of the user.

Both of these functions must be recycled for major revisions. A reasonable criterion for a large operating system is that it shall perform continuously

for at least two weeks without a malfunction affecting the user. It will usually be a minimum of three months before such a status is achieved. Needham of Cambridge University says "There are very few bugs in our operating system that weren't put there in the last two weeks".

4. *What Are the Authorities?*

This is a management decision, or it may be delegated by them to the Q/A group. Seldom, or never, should the release responsibility reside with the producing group. In the case of public safety and welfare depending upon the software, perhaps a Q/A group veto should be protected from a management override.

5. *When Is the System Correct?*

For large systems, it has been recognized that the answer is "Probably never". One should reject the interpretation of 100% "mathematical" or "logical" correctness for software engineering purposes, for reasons of statistical frequency of exercise, and the program interaction with the data:

1. A 99.9% correct program is no better than a 99.8% program, if the data are only 80% correct.

2. If the hardware has a logic flaw, but has a superimposed FORTRAN processor which never exercises that feature, or causes it to be exercised, then the combined system may be said to be correct (apart from other possible flaws).

In short, correctness to the software engineer means that a system should do the "proper" thing rather than do exactly the actions that were specified with such imprecise knowledge.

I prefer the following interpretations of correctness:

1. Design correctness: efficiently utilizes production resources; efficiently utilizes system resources during running; maintainable and reliable; constructible; flexible (for change and added function).

2. Implementation correctness: matches the specifications; solves the problem envisioned; free from malfunction; free from hang-up or locking.

For those that may feel dissatisfied with this thesis, I quote Schorr of IBM, speaking at the 1969 October NATO Conference on Software Engineering: "Apollo 11 software acceptance testing took about two months, and it was at least 30 days before anything would even start to run in real time. Bugs were taken out of the software up until the day before launch".

Thus, we see that system planning for incorrectness is far more effective than excessive emphasis upon absolute correctness that cannot be achieved in finite periods of time.

VII. How Should the Product Be Introduced?

1. *What Are the General Requirements?*

The introduction of a software product is dependent upon the constitution of the product, which may include: (1) The working software, or the wherewithal to generate the working software; (2) operating instructions and rules; (3) technical documentation on data forms, source, actions, flowcharts, and all the other elements normally associated with program transferability; (4) system support (if from an external supplier).

2. *Is It a New Product, Not Replacing an Old One?*

In this case, the main problem is the effect it may have upon data and data file structure used by other programs. It may be completely independent of the action of other programs, and yet have strong interactions with common data.

It is desirable to have sample runs supplied for duplication in the production environment.

3. *Does It Replace a Previous Product?*

1. If it is an update, performing basically the same functions, the main requirement is for a period of parallel running with the old program, comparing production answers.

2. If it is a new product, performing similar but not identical functions, there must be an overlap period for phaseout. Whenever possible, the interface to humans should be consistent with the former interface, as in operating modes, messages, etc.

4. *If Data Conversion Is Required?*

Several types of conversion may be required, such as: (1) graphic set content, encoding, and character size; (2) precision and range of numerals; (3) data formats; (4) file content (added, changed, or deleted); (5) file structure; (6) media labeling; (7) physical media formats.

Once-and-for-all conversion is the exception, and it may be advisable to have

it done on a service basis, particularly by an outside supplier. More commonly, the new and old products must coexist until the new one is proven sufficiently. In this case, it may be useful to have separate files for both the old and the new product, with a bidirectional conversion program to verify identity between the two versions at each stage.

VIII. How Should the Product Be Improved and Serviced?

1. *What Types of Improvements Are Possible?*

1. Toward data and program transferability. This may require:

(a) Changes to comply to standards—programming languages, character sets, data structures, media labeling, etc.

(b) Making the data files self-descriptive and identifying; e.g., copying the Data Division of a COBOL program on the data medium.

(c) Keeping programs in source form, without patches. If patches must be employed, recover source form promptly and periodically.

(d) Making the program self-documenting.

(e) Improvement in ease of use, attention to human factors.

2. Additional capabilities, such as:

(a) New functions or features not previously available.

(b) Functions or features of existing programs which may be taken over, obviating need for those programs.

(c) More choices of algorithms, for better efficiency in alternative situations. This is a particular requirement for basic software, which is optimized for a hypothetical, often nonexistent, user. The user should be provided with instrumentation software and generators for specialization.

3. Better performance, via: (a) instrumentation, (b) design analysis, and (c) restructuring data files.

4. More reliability, via: (a) elimination of hang-up conditions, and (b) confidence and range testing, checking for reasonableness.

2. *Who Should Service the Product?*

1. Trainees? This is usually thought to be a good method of indoctrination and gaining of experience. However, there are severe drawbacks. The trainee can pick up bad habits, will get bored and discouraged easily, takes excessive time to correct malfunctions, and may disrupt other parts of the program in the process of making a specific symptom disappear.

2. Experienced support personnel? It is rare to find programmers who will be happy on a steady diet of correcting other people's mistakes. Forcing them to continue in this function for long time periods leads to job dissatisfaction and resignations.

3. The originator? Why not? He should know it best, and he doesn't necessarily spend all of his time in the support, if other attractive duties are made available to him. If he considers it a trap, let him know that nothing but excellent and self-explaining documentation will release him; he himself can construct the key for release. If the product is substantial and used on a customer site (such as a major basic system for a computer line), it could be desirable to put originators at these sites to keep up to date on field experience. They can still participate in new software production via remote terminals.

IX. Conclusion

Software engineering is in a crisis of identity and maturation, and this has and will lead to promotion of various panaceas, justified by saying that nothing similar has existed before. Not so. We need to use our present tools under good management practices more than we need new and spectacular developments, many of which do not pay off. Art must be reduced to engineering, and software made visible to management in order to avoid the present high spoilage and nontransferability rates. The most profit lies in tooling for production, building new systems via old systems which are stable and mature, instrumenting for effectiveness, and standardizing to make user-developed software reusable and to reduce needless variety.

REFERENCES

1. Garrity, J., "Getting the Most out of Your Computer." McKinsey, New York, 1963.
2. News article, "Software Classification Undertaken as Pilot Project." *Computerworld* p. 2, 1969 November 5.
3. David, E. E., "Some thoughts about the production of large software systems." *NATO Conf. Software Eng., Garmisch, 1968*, excerpted in the *Rep. on Conf., 1969*. NATO, Brussels.
4. Anonymous, "The end of OS." *Datamation*, 14, 72 (1968 April).
5. Conway, M. E., "On the economics of the software market." *Datamation* 14, 28-31 (1968 October).
6. Aron, J. D., "Estimating resources for large programming systems." *2nd NATO Conf. Software Eng., Rome, 1969*. NATO, Brussels.
7. Sackman, H., Erikson, W. J., and Grant, E. E., "Exploratory experimental studies comparing online and offline programming performance." *Comm. ACM* 11, 3-10 (1968).
8. Conway, M. E., "How do committees invent?" *Datamation* 14, 28-31 (1968 April).
9. Bemert, R. W., and Ellison, A. L., "Software instrumentation systems for optimum performance." *Proc. Int. Federation Information Processing Congr. 1968*, Software 2, Booklet C. North-Holland Publ., Amsterdam.

10. Cantrell, H. N., and Ellison, A. L., "Multiprogramming system performance measurement and analysis." *Proc. AFIPS Spring Joint Comput. Conf.*, 1968, pp. 213-223. Thompson, Washington, D.C., 1968.
11. Campbell, D. J., and Hefner, W. J., "Measurement and analysis of large operating systems during system development." *Proc. AFIPS Fall Joint Comput. Conf.*, 1968, Pt. I, pp. 903-914. Thompson, Washington, D.C., 1968.
12. Calingaert, P., "System performance evaluation: Survey and appraisal," *Comm. ACM* 10, 12-18 (1967); (see also associated references provided by Estrin in Review 11661, *CR* 8, 159-160 (1967).

the five areas involved—management, systems, programming, operations and users? In evaluating a system we ask again, does it provide all of the elements of good documentation previously mentioned? Is it available? Is it usable, directly by the persons involved? Does it have good quality—is it current, accurate, clear, objective, reliable, valid? Is it complete? Is it standardized, or is there further room for increasing efficiency? And finally, is it well suited to the intended purpose with the correct level of detail, the correct organization, function and relevancy?

But the evaluation must not end there. For there is a need for a continuing review and maintenance of the new system is to have lasting success. There should be a permanent assigned responsibility, not on the part of the original project team, but should have been reassigned after cutover. Nevertheless, there should, at any point in time, be a designated person known to all those concerned as being responsible for accepting suggestions and coordinating corrections and improvements in the system.

It is highly appropriate to continue to systematically consider input from all quarters. It is necessary to have continuing management support and enforcement from the very bottom of line supervision on up. It is necessary to have continuing education and training, both the new people in the organization as well as the old. And, of course, it is necessary to use the

normal good sound management practices—communication, discipline, etc.—in the implementation and continual effective utilization of this program as with any other.

SUMMARY

In this presentation we addressed first of all just what documentation is, stating that it is essentially anything in written fashion that is used for communication. The elements of good documentation were identified. The justification, the purpose and the objectives were discussed. We said that economics is the primary consideration and good communication the primary use of sound documentation. The contents of a good documentation program were reviewed and some examples that can be considered a method for implementing a good documentation program, treating it as part of a more general standards program and conducting the implementation as if this were just another standard project with all of the attendant methods and controls normally used by good management in conducting such a project. And finally, in evaluating the installed documentation system, the question was asked, "Does the system meet the prescribed objectives and does it do the job effectively?" Of course, it is necessary to continually upgrade, review and maintain the program so that it will have lasting success in its usefulness to the organization. ■

all concerned with standards more than we realize; they are all around us, serving quietly and for the most part unnoticeably, except when one gets in trouble for lack of them.

Let's look at a few examples. Perhaps you have had correspondence from Europe, and have noted the odd size of the paper. I get quite a lot of it and have to keep a cutting board in my office to trim it to fit our ring binders, or to reproduce it to send around the company. Since it is 7/10 of an inch longer than U.S. paper, and the European secretaries type as close to the bottom of the page as do our own, it is often quite difficult not to trim off some copy. So our secretaries complain about this paper, which is called ISO A4 (ISO is the International Standardization Organization, based in Geneva, Switzerland). Why don't the Europeans use the standard size, they ask?

What makes secretaries think that 8-1/2 in. by 11 in. is standard? It isn't to the U.S. Government. For them it is 8 in. by 10-1/2 in., set by law. The British have had a still different size, but now they are going to A4 size in their metrication program. The ISO size has a very sound basis, if we look at the problem of photoreduction or enlargement, or paper stock cutting. This is something that secretaries can relate to, for the U.S. paper size has always given them problems in this matter.

The reason is apparent if one cuts or folds a piece of 8-1/2 in. by 11 in. paper in half horizontally. Turn it vertically and you will notice that the ratio (aspect ratio) of width to height for the half sheet is not the same as for the full sheet (5-1/2 in. to 8-1/2 in. + 8-1/2 in. to 11 in.). It is therefore not projective (as for photoreduction by camera), and that is where the difficulty lies. The ISO A4 size, when folded, is projective, for the simple reason that the height of 297 mm is $\sqrt{2}$ times the width of 210 mm.

Perhaps, like the metric system, this is so logical that we should change. All we would have to do is replace or modify all of the office equipment in this country — like hole punches, ring binders, briefcases, bookshelves, file folders, file drawers, etc. Would such a small difference (easily handled by paper cutters) have any effect upon international computer usage? Absolutely. In a former position as coordi-

Information Processing Standards

by R. W. Bemer
General Electric Co.

Even if one thinks that the topic of standardization is dull and useless, standards can be important, as the owners of a German ocean liner believe (from a news story in the *New York Times*, September 13, 1966):

"Inquiry Studies Hanseatic Fire — The city's fireboats are not equipped with the so-called international hose connections, which the liner had, and therefore could not provide water to the liner's firefighting system, including both hoses and sprinklers, a witness said . . ."

Perhaps the New York City Fire

Commissioners had taken the commonplace attitude described in the August 26, 1968, issue of the *American Machinist Magazine*, asking if the reader's definition of a standard was: "A dull document produced by a committee of dull people who argue interminably and consume reams of paper in letter ballots before they produce a consensus on a position that is already obsolete when it is adopted."

The preferred definition is that of Dr. A. V. Astin, former Director of the National Bureau of Standards:

"A standard is an arbitrary solution to a recurring problem."

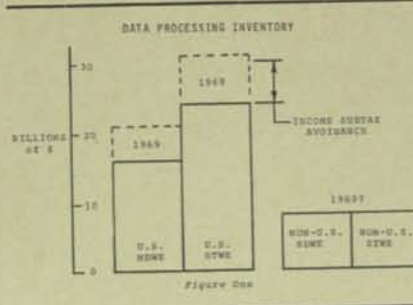
Standardization is not really a dull topic; it just seems that way. We are

nator of systems engineering (and standards), I found a non-impact printer for computer output very nearly in production. Unlike the present impact printers with the wide sheets, this device produced normal page sizes, cut from a continuous roll of paper. I asked what the maximum length of cut was, and the reply was 11 inches. The company confirmed that they planned to market it in Europe. I am afraid that my explanation at that time was not gentle, for the maximum length was built into the physical frame and was not possible to increase.

One international standardization topic should be of considerable interest to the DPMA, for it has partial impetus from computer usage. That is the way one writes the date. In Europe (and formerly in the U.S. Armed Forces) it is written as day-month-year. Much of the U.S. public uses month-day-comma-year. Perhaps this doesn't seem earthshaking, but I was once almost unable to attend an important conference in Europe because I could not get into the country with a smallpox vaccination certificate that expired on 9/3/59, and it was already June!

The ISO has agreed, after eight years, on the Swedish proposal for an ordering of year-month-day. The U.S. Department of Defense has adopted this method effective the first of 1970. The American Bankers Association has made no move as yet to make the forms of checks conform to this change, but that could have a tremendous influence on adoption by the public. Perhaps they remember the public outcries about MICR digits, and the vocal rebellion when all-digit numbers were introduced by the telephone companies. To convert the general public to writing the date in this form will take considerable public relations work.

If data processing people will use this format, as specified in document X3/202 of the American National Standards Institute, they will find some interesting savings in computation time. For example, a single subtraction will tell which of two dates is the earlier. General Electric uses the same principle for scheduling on a fiscal week basis, for project control and PERT charts, i.e., 7046 - 7032 is a 14 week difference. This form is also perfect for ordering (sorting) by date. A companion standard gives



Monday as the first day of the fiscal week. Obviously we would have run into strong religious opposition if this had been generalized to more than business usage, for ISO standards are for everything, not just computers.

There are many reasons to be cautious with respect to standardization. Some of these are:

(1) Be careful of the way in which standards are written. They usually give necessary conditions, but these are not always sufficient. Don't presume anything if you don't have to. For example, the British Standards Institution was drafting a standard for electric typewriters in 1960. I made some comments and requests in behalf of IBM. The draft said that the plane of the keyboard would be between 11 and 16 degrees; I asked if that could read "the plane defined by the top and bottom row." They found no reason to object and agreed, but could see no reason why. The draft said that the diameter of the keytop was 9/16 in.; I asked if that could read "the diameter of the finger contact." Again the same result. Then IBM came out with the Selectric, which has a concave keyboard and no keytops as such.

(2) Don't believe things too abjectly, or accept them as obvious. I once gave a paper on program transferability, wherein I said that on the whole it was a healthy phenomenon to get different answers from the same program running on a different computer. One of my co-workers objected to this statement, so I gave him an example:

"The U.S. Army had run a FORTRAN object program on a 7090 for three years. UNIVAC was attempting to sell an 1107. In compiling the source program, a diagnostic message said that there was an entry to the middle of a DO loop, which had not been recognized for those three years of wrong answers."

(3) Don't think that some things are too simple and trivial to be

bothered with. Take the example of the COBOL statement:

```
IF CHARACTER EXCEEDS 'S'
THEN NEXT STATEMENT
OTHERWISE STOP
```

Unfortunately the IBM 360 COBOL will give the opposite action from that of the NCR Century COBOL. For this reason, and also because the U.S. Government has made such a directive for file representations, we have now persuaded CODASYL to adopt the ASCII collating sequence for COBOL. But watch out, as this will give spurious solutions for two-case usage.

Going from specifics to the general, there are many reasons for standardization in data and information processing. Some of these are:

- Data interchange and movement
- Multiple use of data (banks)
- Transfer of data problem solutions (programs) and documentation to:
 - Additional equipment
 - Multiple equipment
 - Backup equipment
 - Linked equipment
 - New equipment
 - Different equipment
- Economy of competitive acquisition (interfaces, mixed systems)
- Capture of other work, avoidance of reinvention
- Flexibility in response to changing requirements
- Personnel turnover and training

We can give many more, but they all come down to one thing - money! We are playing in a big game with big dollars, as Figure 1 shows. Accompanying over \$5 billion in hardware in 1969 was about \$7 billion in software and mechanically recorded data. A major redundancy factor ex-

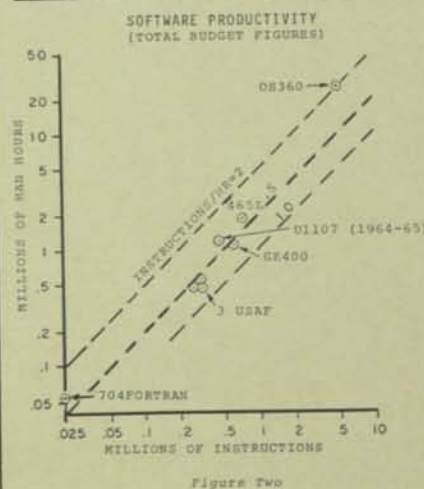
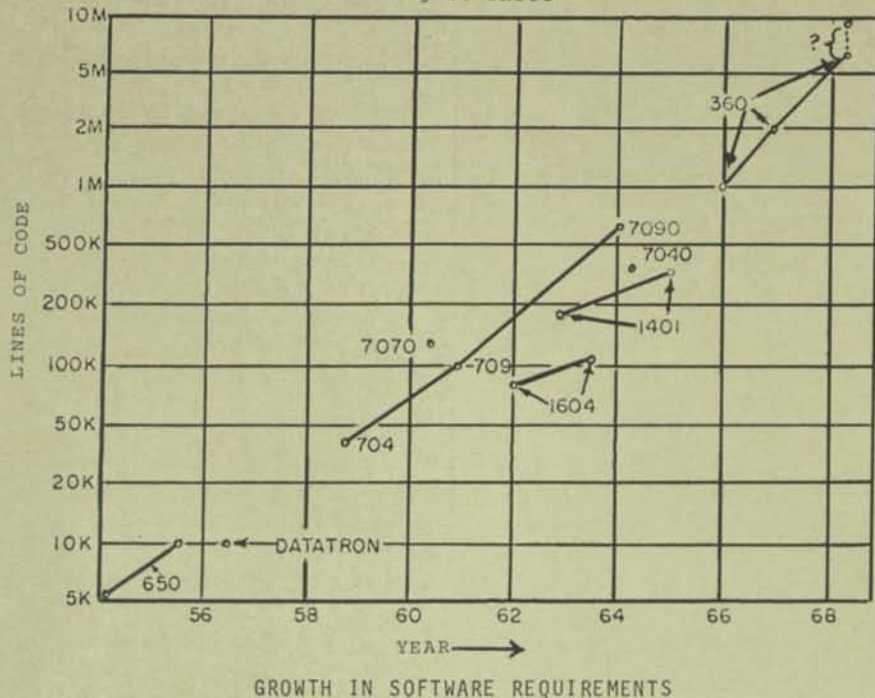


Figure Three



GROWTH IN SOFTWARE REQUIREMENTS

ists, however, when looking at actual and projected figures. The U.S. Government gets from 30 to 50% utilization from their equipment, other users not much more, and they worry about it. But how about that \$7 billion in software? No more than \$1 billion worth is reusable on other equipment and other people's problems, due to transferability problems. This is an even lower utilization figure.

Perhaps this waste can be avoided by some new miracle. Figure 2 shows how the miracles are coming. This chart is designed to reflect total budget figures on the basis of approximately 30 per cent for design and implementation, 20 per cent for test and 50 per cent for management, documentation and support. It is related to the McClure chart on the size of basic software systems (Figure 3). Using these charts one may extrapolate to 1972 to find a computer system with 25 million instruction software, costing \$1.25 billion, constructed by 15,000 programmers. Something has to be done about the difficulties of transferability. Standards are a substantial part of the answer, and that wasted \$6 billion a year tells me that they are very important.

Of course we can always get along with the crutch of emulation. We can argue that differences must be perpetuated because it costs too much to change. This is why the U.S. still has not gone metric, yet it costs more

each year in waste, and will cost even more to make the inevitable change. See Figure 4.

One of my friends at IBM tried this in the Spring of 1969, except that he put the 702 inside the 705. He reported that the program ran slightly faster on the 360 than it did on the 702, vintage 1954. It is shocking how many people are fooling themselves and running like this. Many do not even use their files in EBCDIC, but rather the old 6-bit code of the 705! It was said that 80 per cent of the 7080s themselves were run with the switch at the 705 MOD I position. Why can't we move to use new equipment at its best? Is it the program or the data that causes the difficulties? I have a little saying that "If the data are not transferable — the program cannot be transferable".

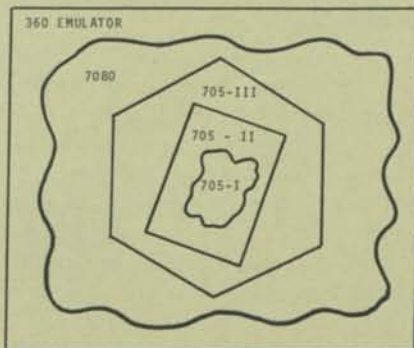


Figure Four

One of the present difficulties in data processing standardization is that we are still working on standards in the areas indicated by the past decade, not in the area of greatest opportunity and payoff in the next decade. The difference may be illustrated by starting with the following definitions:

Data — A representation of facts or ideals in a formalized manner capable of being communicated or manipulated by some process.

Information — The meaning that a human assigns to data by means of the known conventions used in its representation.

The distinction can also be made on the basis that if you can move it, put it away, find it again, transform it and untransform it — without knowing what it meant — it's data!

In the sixties we processed more information than data; in the seventies the processing of data will outweigh by far the processing of information (when the content is changed in any way). The reason for this is, of course, that we shall have more need for simple access, display and data movement — as computers are integrated more directly into human activities. Even now it is difficult to awaken the standardization people to the importance of data — its structure and elements. We are going to have to look at computing as it will be, not as it was. Programmers have been concerned for too many years with algorithms and programming languages. An algorithm is primarily an information process performed upon data. In the past these data have been relatively homogenous and from close or related sources.

Moving to data banks we must consider anew these processes, for the data are no longer necessarily homogenous in structure, nor are they necessarily from related sources. Data will now have to be public in nature; this does not mean free from safeguards of privacy and security, but rather that it can be used by all who have a right to access. The difficulty with our present inventory of mechanically-recorded data is that it is essentially local and private data, hampered by information losses that pre-

vent it from going public. Making public data private is relatively easy; one withdraws it or puts legislative or other controls upon its usage. However, recovering from those information losses to make private data public is unbelievably difficult.

The real purpose of data processing is to have the program and data dance together. One may dance marathon style, or periodically with long and short intermissions. In private a single couple may dance as they please to their own music source, but in public there are constraints as to when the hall and the orchestra are available. This is where operating systems come in. They provide the time, place and facilities for the data and program to dance, as it were. Now communications and data banks make it possible for the same data to dance in many ballrooms, even simultaneously, and with different program partners.

To do this at all efficiently (for reasons of data transferability and reuse) it is necessary to make the data management system the highest in the hierarchy, as noted in Figure 5. Operating systems are subservient, and there may be different operating systems associated with a single data management system, each providing the ballroom for their programs to interact with the data.

If data dances in many ballrooms there is going to be a recognition problem. Thus data must be identified as to type — either by data descriptive language or by identification to allow one to look somewhere for the characteristics. Thus there is a need for levels of identification and familiarity, as well as for levels of privacy. There is a rather universal mechanism to accomplish these, known as "ESCAPE". ESCAPE usually indicates a registered sequence which gives the identification number of a different character set, or variations in media labels, data formats and data communications control procedures. It may extend infinitely, for one can escape to another escape domain.

If one accepts my argument of the separation of the data base management system from the operating system, even though the ultimate benefit is not so apparent now, then it will be seen that there are many things wrong with our existing standards. For example, the Data Division of COBOL is a part of the program, not of the data tape or other media. De-

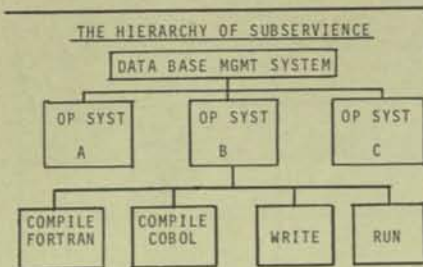


Figure Five

stroy the program and what is on the tape? Furthermore, the data procedures are not common between COBOL, PL/I and FORTRAN. There is no reason why they shouldn't be common, and the users are paying for this in operating inefficiency and unnecessary software using up valuable storage.

I do not wish to emphasize standards of compliance more than standards of performance. Both contribute heavily to the efficiency and cost-performance effectiveness of computer utilization. In both areas, however, I am at a great disadvantage to convince users of the relative value of standards, and to ask the users' support in their creation and adoption, because many users cannot relate to what I am saying, lacking quantitative tools to measure the cost to performance of lack or misuse of standards.

If some action takes 10 milliseconds that should take only one, the human cannot detect it in his software system, nor can he relate to it without measurement. When we instrumented the 600 software (a first in the industry) we found some serious system inefficiencies. Correction has enabled the improvement of performance by

better than two-to-one. Some firms now supply instrumentation for customer programs, and have demonstrated 20 to 40 per cent performance improvement in a short test time. But these are primarily for user's programs, not the manufacturer- or software house-supplied basic software. The operation of this software is pretty much out of the user's control, and very likely he is paying heavily (and unwittingly) for processes which are either useless or inefficient.

Standards affect the inefficient processes, such as conversion to and from the ISO (ASCII) Code in communication-based systems. Standardization workers would find it easier to talk to users on an understanding basis if the users could only find out from their computer salesman which elements of the hardware and software system were there to get around non-standardization, and then add up the cost.

In closing there are two excellent sources of such information detailing the actual and diverse standardization activities:

(1) The series of notes on Federal Information Processing Standards, from the Center of Computer Sciences and Technology, the National Bureau of Standards. These are available in the NBS Technical News Bulletin from the Supt. of Documents, U.S. Government Printing Office, Washington, D.C. A yearly subscription costs \$3.

(2) BEMA, the Business Equipment Manufacturers Association, puts out a quarterly progress report on national and international standardization for computers and information processing. Available upon request. ■

Standardization—What, Why and How?

by Milt Bryce
President
TekFax, Inc.

In discussing our attitudes toward standards, I suggest that the reason we, as a profession, have not developed a formalized body of standards is that we have emotional hang-ups on the subject. Some individuals have developed standards and some installations have standards, but why as a group have we resisted attacking this problem head-on? Why, 19 years after the first commercial computer was an-

nounced, are we still wrestling with standards?

One reason might be that such things as standards and standard operating procedures fail to fit the image some of us have of ourselves. We like impressing others with our computer gibberish. If we became business-like and used standards, we might lose this image; the mystique of the computer room might disappear.

Also, standards are, by definition, a measure or a base for comparison purposes. Are we afraid of having our performance measured? I have a feel-

INFORMATION PROCESSING STANDARDS

R. W. Bemer, the General Electric Co., Phoenix, Arizona

LADIES AND GENTLEMEN, I AM HERE TO TRY TO WHIP UP A LITTLE ENTHUSIASM ON A TOPIC THAT MANY THINK IS DULL AND USELESS. EVEN IF DULL, STANDARDS CAN BE IMPORTANT, AS THE OWNERS OF A GERMAN OCEAN LINER BELIEVE:

① (SLIDE - INQUIRY STUDIES HANSEATIC FIRE)

PERHAPS THE NEW YORK CITY FIRE COMMISSIONERS HAD TAKEN A COMMONPLACE ATTITUDE ABOUT STANDARDS:

② (SLIDE - IS THIS YOUR DEFINITION OF A STANDARD?)

I DON'T THINK STANDARDIZATION IS REALLY A DULL TOPIC. I THINK THAT SO FAR WE'VE HAD MOSTLY DULL PEOPLE EXPLAINING IT, WHO DO NOT HAVE ANY AWARENESS OF THE NEED FOR PUBLIC RELATIONS, OR THE DESIRE TO MAKE IT INTERESTING. I HOPE THAT I CAN MAKE A CONTRIBUTION HERE, FOR YOU ARE ALL CONCERNED WITH STANDARDS MORE THAN YOU KNOW. THEY ARE ALL AROUND US, SERVING QUIETLY AND FOR THE MOST PART UNNOTICEABLY, EXCEPT WHEN ONE GETS IN TROUBLE FOR LACK OF THEM. THE SWEDISH STANDARDS COMMISSION GIVES OUT MATCHBOOKS WITH A CARTOON OF A PARTLY DRESSED LADY COVERING HERSELF IN PANIC BECAUSE HER WINDOW SHADE IS TOO SMALL FOR THE WINDOW.

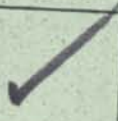
LET'S LOOK AT A FEW EXAMPLES. PERHAPS YOU HAVE HAD CORRESPONDENCE FROM EUROPE, AND HAVE NOTED THE ODD SIZE OF THE PAPER. I GET QUITE A LOT IN MY JOB AND HAVE TO KEEP A CUTTING BOARD IN MY OFFICE TO TRIM IT TO FIT OUR RING BINDERS, OR TO REPRODUCE IT TO SEND AROUND THE COMPANY. SINCE IT IS 7/10 OF AN INCH LONGER THAN U.S. PAPER, AND THE EUROPEAN SECRETARIES TYPE TO AS CLOSE TO THE BOTTOM OF THE PAGE AS DO OUR OWN, IT IS OFTEN QUITE DIFFICULT NOT TO TRIM OFF SOME COPY. SO OUR SECRETARIES COMPLAIN ABOUT THIS PAPER, WHICH IS CALLED ISO A4 (ISO IS THE INTERNATIONAL STANDARDIZATION ORGANIZATION, BASED IN GENEVA, SWITZERLAND). WHY DON'T THE EUROPEANS USE THE STANDARD SIZE, THEY ASK?

SO WHAT MAKES THEM THINK THAT 8 1/2 BY 11 IS STANDARD? IT ISN'T TO THE U.S. GOVERNMENT. FOR THEM IT IS 8 BY 10 1/2, BY LAW! THE BRITISH HAVE HAD A STILL DIFFERENT SIZE, BUT NOW THEY ARE GOING TO A4 IN THEIR METRICATION PROGRAM. IS ISO CRAZY? LET'S LOOK AT THE PROBLEM OF PHOTOREDUCTION OR ENLARGEMENT, OR PAPER STOCK CUTTING. HERE WE HAVE SOMETHING SECRETARIES UNDERSTAND, FOR THE U.S. PAPER HAS ALWAYS GIVEN THEM PROBLEMS IN THIS MATTER. HERE'S WHY.

(DEMONSTRATION - FOLDED PAPER ON DIAGONAL, U.S. AND A4)

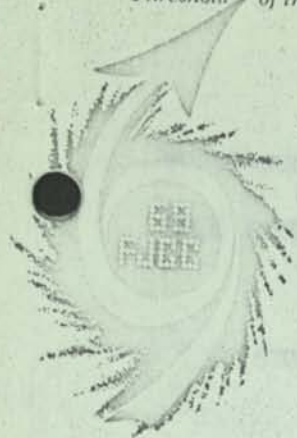
③ (SLIDE - THE MAGIC OF $\sqrt{2}$)

PERHAPS, LIKE THE METRIC SYSTEM, THIS IS SO LOGICAL THAT WE SHOULD CHANGE? ALL WE WOULD HAVE TO DO IS REPLACE OR MODIFY ALL OF THE OFFICE EQUIPMENT IN THIS COUNTRY--LIKE HOLE PUNCHES, RING BINDERS, BRIEFCASES, BOOKSHELVES, FILE FOLDERS, FILE DRAWERS, ETC. YOU CAN PROBABLY THINK OF MANY OTHER ITEMS IN THE "ETC." CLASS.



FALL JOINT COMPUTER CONFERENCE

MANDATORY VISUAL LAYOUT FORM



INSTRUCTIONS: This frame, and the specified symbol size, when sealed up to 6 x 8 feet will accommodate a four- to five-hundred man audience. Projection screen size constraints of this order, and audiences of this size, are routinely encountered at the FJCC, and speaker's visuals must be standardized accordingly.

Print in capitals to the full height of the numbered blue-grid lines, not going beyond the blue-grid margins. First two lines are to be used for thesis titles. Alternatively, material may be typed, all caps, double-spaced, preferably using a machine with a carbon ribbon and

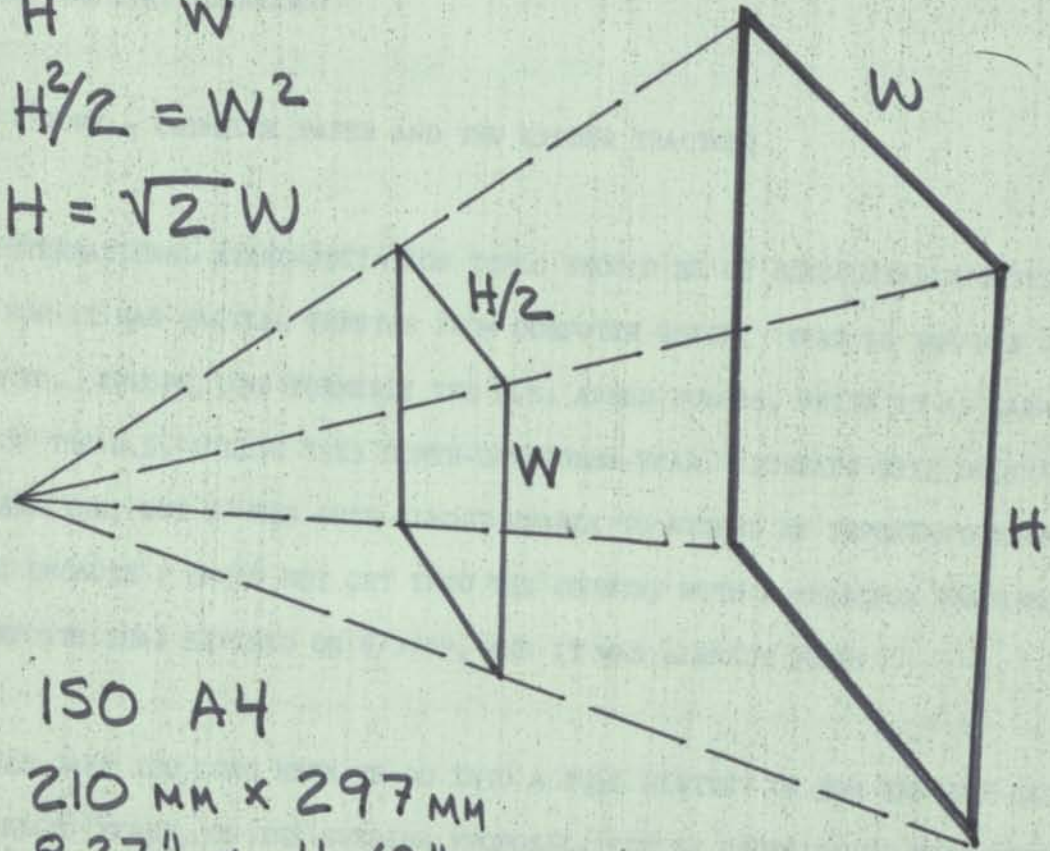
a type size equal to or larger than the standard "elite." If typed, do not go beyond the *inner* double-lined blue margins.

The completed frame may be copied as a 35-mm slide (mandatory size for the FJCC), used directly to make a small Thermofax or Xerox transparency for an overhead projector for rehearsals, or used to scale rough art for professional rendering. The frames, when pinned on a wall in sequence (storyboarded), provide the best possible device for preliminary appraisal of the continuity and impact of a technical presentation.

$$\frac{W}{H} = \frac{H/2}{W}$$

$$H^2/2 = W^2$$

$$H = \sqrt{2} W$$



ISO A4
 210 MM x 297 MM
 8.27" x 11.69"

MOVE TO RIGHT 1/2"
 (OR MAKE FOUR DOTS 1/2" LEFT)

WOULD SUCH A SILLY DIFFERENCE (EASILY HANDLED BY PAPER CUTTERS) HAVE ANY EFFECT UPON INTERNATIONAL COMPUTER USAGE?

(STORY - OMNITRONIC PRINTER, TO SELL IN EUROPE)

OF COURSE, OUR PRINTER DESIGNERS IN EUROPE HAVE MUCH THE SAME TYPE OF PROBLEM FOR THE U.S. MARKET, JUST BECAUSE WE DO NOT YET HAVE STANDARDS IN THE PAPER WE USE FOR LINE PRINTERS!

(STORY - CRIMPLOK PAPER AND THE KIDDER TRACTOR)

ONE INTERNATIONAL STANDARDIZATION TOPIC SHOULD BE OF CONSIDERABLE INTEREST TO YOU, FOR IT HAS PARTIAL IMPETUS FROM COMPUTER USAGE. THAT IS THE WAY ONE WRITES THE DATE. EUROPE, AND FORMERLY THE U.S. ARMED FORCES, WRITE IT AS DAY-MONTH-YEAR. MUCH OF THE U.S. PUBLIC USES MONTH-DAY-COMMA-YEAR. PERHAPS THIS DOESN'T SEEM EARTHSHAKING, BUT I WAS ONCE ALMOST UNABLE TO ATTEND AN IMPORTANT CONFERENCE IN EUROPE BECAUSE I COULD NOT GET INTO THE COUNTRY WITH A SMALLPOX VACCINATION CERTIFICATE THAT EXPIRED ON 9/3/59, AND IT WAS ALREADY JUNE!

IT WOULD TAKE TOO LONG HERE TO GO INTO A FULL HISTORY OF HOW THE ISO HAS AGREED, AFTER EIGHT YEARS, ON THE SWEDISH PROPOSAL, FOR AN ORDERING OF YEAR-MONTH-DAY. THE U.S. DEPARTMENT OF DEFENSE HAS ADOPTED THIS METHOD EFFECTIVE THE FIRST OF THIS PRESENT YEAR. THE AMERICAN BANKERS ASSOCIATION HAS MADE NO MOVE AS YET TO MAKE THE FORMS OF CHECKS FOR THIS ORDERING, BUT THAT COULD HAVE A TREMENDOUS INFLUENCE ON ADOPTION BY THE PUBLIC. PERHAPS THEY REMEMBER THE PUBLIC OUTCRIES ABOUT MICR DIGITS, AND THE VOCAL REBELLION WHEN ALL-DIGIT NUMBERS WERE

INTRODUCED BY THE TELEPHONE COMPANIES. TO CONVERT THE GENERAL PUBLIC TO WRITING THE DATE IN THIS FORM WILL TAKE CONSIDERABLE P.R.

BE CAREFUL OF HOW STAMPS ARE WRITTEN, THEY ARE USUALLY NECESSARY, BUT AS AN EXAMPLE OF REACTION, SOME PEOPLE COMPLAIN THAT IT IS TOO DIFFICULT TO RELEARN, AND WHAT ABOUT ALL THOSE DATE STAMPS IN OFFICES? WELL, I HAVE USED THIS ORDERING FOR SIX YEARS NOW; I HAVE THE CHANGE TIME DOWN TO FIVE MINUTES FOR TWO WELL-KNOWN MODELS OF DATE STAMPS, AND SECRETARIES CAN CONVERT OVERNIGHT, ONCE I RUN THROUGH A SIMPLE TEST. I ASK THE TIME ON MY WATCH, WITHOUT USING ANY PREPOSITIONS IN THE REPLY, THEN I ASK "OH - YOU MEAN YOU MENTION THE LARGER UNIT FIRST?"

IF YOU DATA PROCESSING PEOPLE WILL USE THIS FORMAT, AS SPECIFIED IN DOCUMENT X3/202 OF THE AMERICAN NATIONAL STANDARDS INSTITUTE, YOU WILL FIND THAT THERE ARE SOME INTERESTING SAVINGS IN COMPUTATION TIME. FOR EXAMPLE, A SINGLE SUBTRACTION WILL TELL WHICH OF TWO DATES IS THE EARLIER. GENERAL ELECTRIC USES THE SAME PRINCIPLE FOR SCHEDULING ON A FISCAL WEEK BASIS, FOR PROJECT CONTROL AND PERT CHARTS. 7046 - 7032 IS A FOURTEEN WEEK DIFFERENCE. PERFECT FOR ORDERING (SORTING) BY DATE, TOO. I REMEMBER WHEN THE IBM 705 SORT COULD NOT HANDLE THE OLD DATE FORM--IT WOULD TAKE ONLY FIVE KEY ELEMENTS. AS AN ASIDE, A COMPANION STANDARD TO THIS GIVES MONDAY AS THE FIRST DAY OF THE FISCAL WEEK. OBVIOUSLY WE WOULD HAVE RUN INTO STRONG RELIGIOUS OPPOSITION IF THIS HAD BEEN GENERALIZED TO MORE THAN BUSINESS USAGE.

LET ME GIVE YOU THREE WAYS TO BE CAUTIOUS WITH RESPECT TO STANDARDIZATION:

1. BE CAREFUL OF HOW STANDARDS ARE WRITTEN. THEY ARE USUALLY NECESSARY, BUT NOT ALWAYS SUFFICIENT. DON'T PRESUME IF YOU DON'T HAVE TO.

(STORY - BSI IN 1960. PLANE OF KEYBOARD, 9/16" KEYPAD. THEN THE IBM SELECTRIC.)

2. DON'T BELIEVE THINGS TOO ABJECTLY, OR ACCEPT THEM AS OBVIOUS. I ONCE GAVE A PAPER ON PROGRAM TRANSFERABILITY, AND SAID THAT ON THE WHOLE IT WAS A HEALTHY PHENOMENON TO GET DIFFERENT ANSWERS FROM THE SAME PROGRAM RUNNING ON A DIFFERENT COMPUTER. ONE OF MY CO-WORKERS OBJECTED TO THIS STATEMENT, SO I FOUND HIM SOME EXAMPLES:

"THE U.S. ARMY HAD RUN A FORTRAN OBJECT PROGRAM ON A 7090 FOR THREE YEARS. UNIVAC WAS ATTEMPTING TO SELL AN 1107. IN COMPILING THE SOURCE PROGRAM, A DIAGNOSTIC MESSAGE SAID THAT THERE WAS AN ENTRY TO THE MIDDLE OF A DO LOOP, WHICH HAD NOT BEEN RECOGNIZED FOR THOSE THREE YEARS OF WRONG ANSWERS."

"A LARGE MATRIX WAS BEING INVERTED IN SHORT (32-BIT WORD) PRECISION. THE PROGRAM WAS THEN MOVED TO A 48-BIT WORD MACHINE. THE USER THOUGHT HE HAD 5-DECIMAL ACCURACY IN THE ANSWERS, WAS MAKING DECISIONS BASED UPON 3 DECIMAL DIGITS, AND NOW FOUND OUT THAT IT WASN'T ANY BETTER THAN ONE DIGIT."

3. DON'T THINK THAT SOME THINGS ARE TOO SIMPLE TO BE BOTHERED WITH.

④ (SLIDE - IF CHARACTER EXCEEDS 'S'.)

WE HAVE NOW PERSUADED CODASYL TO ADOPT THE ASCII COLLATING SEQUENCE FOR COBOL BECAUSE THE IBM 360 AND THE NCR CENTURY ACT DIFFERENTLY FOR THIS STATEMENT, AND ALSO BECAUSE THE U.S. GOVERNMENT HAS MADE SUCH A DIRECTIVE FOR FILE REPRESENTATIONS. BUT WATCH OUT FOR SPURIOUS SOLUTIONS TO THE TWO-CASE PROBLEM. IF YOU USE THE STRAIGHT COLLATING SEQUENCE FOR TELEPHONE BOOKS YOU WILL GET SOME ANGUISHED SUBSCRIBERS:

⑤ (SLIDE - DIRECTORY EXCERPT)

NOW LET'S GET DOWN TO BASICS. THERE ARE MANY REASONS FOR STANDARDIZATION IN DATA AND INFORMATION PROCESSING: SOME OF THESE ARE:

- DATA INTERCHANGE AND MOVEMENT
- MULTIPLE USE OF DATA (BANKS)
- TRANSFER OF DATA, PROBLEM SOLUTIONS (PROGRAMS) AND DOCUMENTATION TO:

ADDITIONAL EQUIPMENT

MULTIPLE "

BACKUP "

LINKED "

NEW "

DIFFERENT "

AND FOR BROKERAGE

THIS SIGN " TO REFLECT TOTAL BUDGET FIGURES ON THE BASIS OF APPROXIMATELY 30% FOR DESIGN AND IMPLEMENTATION, 20% FOR TEST, AND 50% FOR MAINTENANCE, DOCUMENTATION AND SUPPORT.

- o ECONOMY OF COMPETITIVE ACQUISITION (INTERFACES, MIXED SYSTEMS)
- o CAPTURE OF OTHER WORK, AVOIDANCE OF REINVENTION
- o FLEXIBILITY IN RESPONSE TO CHANGING REQUIREMENTS
- o PERSONNEL TURNOVER AND TRAINING

WE CAN GIVE MANY MORE, BUT THEY ALL COME DOWN TO ONE THING--MONEY! WE ALL LIKE IT BECAUSE WESTINGHOUSE, GE, AND SYLVANIA LIGHT BULBS FIT THE SAME SOCKET AND GIVE US A CHEAPER PRICE VIA COMPETITION. BUT WE ARE PLAYING IN A BIGGER GAME THAN LIGHT BULBS. PERHAPS SOME MAY NOT REALIZE HOW BIG:

⑥ (SLIDE - DATA PROCESSING INVENTORY)

WITH OVER \$5 BILLION IN HARDWARE IN 1969 WENT ABOUT \$7 BILLION IN SOFTWARE AND MECHANICALLY RECORDED DATA. OUR BUSINESS IS EXTRAPOLATED TO BE THE LARGEST IN THE COUNTRY SOME TIME AROUND THE END OF THIS NEW DECADE. A MAJOR REDUNDANCY FACTOR EXISTS, HOWEVER. THE U.S. GOVERNMENT GETS FROM 30 TO 50% UTILIZATION FROM THEIR EQUIPMENT, OTHER USERS NOT MUCH MORE, AND THEY WORRY ABOUT IT. BUT HOW ABOUT THAT \$7 BILLION IN SOFTWARE? NO MORE THAN ONE BILLION DOLLARS WORTH IS REUSABLE ON OTHER EQUIPMENT AND OTHER PEOPLE'S PROBLEMS, DUE TO TRANSFERABILITY PROBLEMS.

PERHAPS YOU THINK THIS CAN BE AVOIDED BY SOME NEW MIRACLE? LET ME SHOW YOU HOW THE MIRACLES ARE COMING:

⑦ (SLIDE - PRODUCTIVITY OF BASIC SOFTWARE)

THIS CHART IS DESIGNED TO REFLECT TOTAL BUDGET FIGURES ON THE BASIS OF APPROXIMATELY 30% FOR DESIGN AND IMPLEMENTATION, 20% FOR TEST, AND 50% FOR MANAGEMENT, DOCUMENTATION AND SUPPORT.

THIS IS RELATED TO THE MC CLURE CHART ON SIZE OF BASIC SOFTWARE SYSTEMS:

⑧ (SLIDE - GROWTH IN SOFTWARE REQUIREMENTS)

USING THESE CHARTS ONE MAY EXTRAPOLATE TO 1972 TO FIND A COMPUTER SYSTEM WITH 25 MILLION INSTRUCTION SOFTWARE, COSTING \$1.25 BILLION, CONSTRUCTED BY 15,000 PROGRAMMERS. SOMETHING HAS TO BE DONE ABOUT TRANSFERABILITY. STANDARDS ARE A SUBSTANTIAL PART OF THE ANSWER, AND THAT WASTED \$6 BILLION A YEAR TELLS ME THAT THEY ARE VERY IMPORTANT. I HOPE IT TELLS YOU THAT AND GENERATES SOME ACTION. OF COURSE WE CAN ALWAYS GET ALONG WITH THE CRUTCH OF EMULATION. WE CAN ARGUE THAT DIFFERENCES MUST BE PERPETUATED BECAUSE IT COSTS TOO MUCH TO CHANGE. THIS IS WHY THE U.S. STILL HAS NOT GONE METRIC, YET IT COSTS MORE EACH YEAR IN WASTE, AND MORE TO MAKE THE INEVITABLE CHANGE. HERE IS WHAT HAPPENS:

⑨ (SLIDE - NO STANDARDS FOR TRANSFERABILITY)

ONE OF MY FRIENDS AT IBM TRIED THIS IN THE SPRING OF 1969, EXCEPT THAT HE PUT THE 702 INSIDE THE 705. HE REPORTED THAT THE PROGRAM RAN SLIGHTLY FASTER ON THE 360 THAN IT DID ON THE 702, VINTAGE 1954.

IT IS SHOCKING HOW MANY PEOPLE ARE FOOLING THEMSELVES AND RUNNING LIKE THIS. MANY DO NOT EVEN USE THEIR FILES IN EBCDIC, BUT RATHER THE OLD 6-BIT CODE OF THE 705! IT WAS SAID THAT 80% OF THE 7080s THEMSELVES WERE RUN WITH THE SWITCH AT THE 705 MOD I POSITION.

IT'S NOT A LAUGHING MATTER, AS HOWARD SMITH AND I ONCE THOUGHT WHEN WE PLANNED TO HOAX THE INDUSTRY BY PRETENDING TO FIND AN OLD MANUSCRIPT BY COUNTESS LOVELACE, ENTITLED "SIMULATION OF YE DIFFERENCE ENGINE UPON YE ANALYTIC ENGINE".

WHY CAN'T WE MOVE TO USE NEW EQUIPMENT AT ITS BEST? IS IT THE PROGRAM OR THE DATA THAT CAUSES THE DIFFICULTIES?

(b) (SLIDE - IF THE DATA ARE NOT...)

ALREADY WE SEE SIGNS OF SERVICES ARRIVING IN RESPONSE TO THE PROBLEMS OF PROGRAM (AND MORE BASICALLY DATA) TRANSFERABILITY. COMPUTERWORLD HAD AN ARTICLE (1970 FEB.) ON THE FORMATION OF A NEW FIRM.

"COMPUTER CONVERSIONS, INC., INTENDS TO SPECIALIZE IN HELPING FIRMS SURMOUNT CONVERSION PROBLEMS. BELIEVING THAT HUNDREDS, EVEN THOUSANDS, OF COMPUTER INSTALLATIONS ARE NOT ABLE TO GET THE BEST OUT OF NEW TECHNOLOGIES SIMPLY BECAUSE THEY DON'T HAVE ADEQUATE IN-HOUSE CONVERSION CAPABILITIES.

"COMPUTER CONVERSIONS INTENDS TO HELP ITS CLIENTS NOT ONLY IN THE SELECTION OF EQUIPMENT AND NEGOTIATION OF CONTRACTS BUT ALSO IN THE SPECIFIC DEVELOPMENT OF SYSTEMS AND PROCEDURES FOR THE EFFICIENT CONVERSION FROM THE OLD EQUIPMENT TO THE NEW. THIS INCLUDES, WHERE APPROPRIATE, TRAINING OF PROGRAMMING AND OPERATING STAFFS, CONVERSION OF FILES, TESTING, AND DOCUMENTATION OF OPERATIONAL PROGRAMS."

ONE OF OUR PRESENT DIFFICULTIES IN DATA PROCESSING STANDARDIZATION IS THAT WE ARE STILL WORKING ON STANDARDS IN THE AREAS INDICATED BY THE PAST DECADE, NOT IN THE AREA OF GREATEST OPPORTUNITY AND PAYOFF IN THE NEXT DECADE. LET'S LOOK AT THE DIFFERENCE BY STARTING WITH BASIC DEFINITIONS:

(1) (SLIDE - DEFINITION OF "DATA")

(2) (SLIDE - DEFINITION OF "INFORMATION")

(3) (SLIDE - HOW TO RECOGNIZE DATA)

IT IS RATHER INTERESTING THAT THE NAME OF THIS ASSOCIATION IS BECOMING MORE PERTINENT FOR THE NEXT DECADE:

(4) (SLIDE - HOW IT WAS IN 196X)

(5) (SLIDE - HOW IT WILL BE IN 197X) (MORE DISPLAY AND MOVEMENT)

EVEN NOW IT IS DIFFICULT TO AWAKEN THE STANDARDIZATION PEOPLE TO THE IMPORTANCE OF DATA--ITS STRUCTURE AND ELEMENTS. WE ARE GOING TO HAVE TO LOOK AT COMPUTING AS IT WILL BE, NOT AS IT WAS. PROGRAMMERS HAVE BEEN CONCERNED FOR TOO MANY YEARS WITH ALGORITHMS AND PROGRAMMING LANGUAGES. AN ALGORITHM IS PRIMARILY AN INFORMATIONAL PROCESS PERFORMED UPON DATA. IN THE PAST THESE DATA HAVE BEEN RELATIVELY HOMOGENOUS AND FROM CLOSE OR RELATED SOURCES.

MOVING TO DATA BANKS WE MUST CONSIDER ANEW THESE PROCESSES, FOR THE DATA ARE NO LONGER HOMOGENOUS IN STRUCTURE, NOR ARE THEY NECESSARILY FROM RELATED SOURCES:

(6) (SLIDE - CHANGING THE RULES)

THE REAL PURPOSE OF DATA PROCESSING IS TO HAVE THE PROGRAM AND DATA DANCE TOGETHER. ONE MAY DANCE MARATHON STYLE, OR PERIODICALLY WITH LONG AND SHORT INTERMISSIONS. IN PRIVATE A SINGLE COUPLE MAY DANCE AS THEY PLEASE TO THEIR OWN MUSIC SOURCE, BUT IN PUBLIC THERE ARE CONSTRAINTS AS TO WHEN THE HALL AND THE ORCHESTRA ARE AVAILABLE. THIS IS WHERE OPERATING SYSTEMS COME IN. THEY PROVIDE THE TIME, PLACE AND FACILITIES FOR THE DATA AND PROGRAM TO DANCE, AS IT WERE.

WE DID NOT USED TO HAVE TO WORRY TOO MUCH ABOUT HOW TO DANCE WITH STRANGERS. IN THE NINETEENTH CENTURY WE DID NOT TRAVEL ENOUGH TO KNOW MANY STRANGERS. SIMILARLY, DATA AND PROGRAMS HAVE BEEN VERY FAMILIAR TO EACH OTHER. IN FACT, THE STRUCTURE OF THE DATA HAS COMMONLY BEEN BURIED IMPLICITLY IN THE PROGRAM. BUT NOW COMMUNICATIONS AND DATA BANKS MAKE IT POSSIBLE FOR THE SAME DATA TO DANCE IN MANY BALLROOMS, EVEN SIMULTANEOUSLY, AND WITH DIFFERENT PROGRAM PARTNERS.

TO DO THIS AT ALL EFFICIENCY^{TL} (FOR REASONS OF DATA TRANSFERABILITY AND REUSAGE) IT IS NECESSARY TO MAKE THE DATA MANAGEMENT SYSTEM THE HIGHEST IN THE HIERARCHY:

Ⓟ (SLIDE - THE HIERARCHY OF SUBSERVIENCE)

OPERATING SYSTEMS ARE SUBSERVIENT TO IT, AND THERE MAY BE MANY DIFFERENT OPERATING SYSTEMS ASSOCIATED WITH A SINGLE DATA MANAGEMENT SYSTEM, EACH PROVIDING THE BALLROOM FOR THEIR PROGRAMS TO INTERACT WITH THE DATA.

IF DATA DANCES IN MANY BALLROOMS THERE IS GOING TO BE A RECOGNITION PROBLEM.
(DO YOU DANCE SWAHILI OR NEBRASKA CORNHUSKER STYLE?) THUS DATA MUST BE
IDENTIFIED AS TO TYPE--EITHER BY DATA DESCRIPTIVE LANGUAGE OR BY IDENTIFICATION
THAT ONE CAN LOOK UP SOMEWHERE FOR THE CHARACTERISTICS. IT IS JUST LIKE THE
RECOGNITION PROCESS BETWEEN HUMANS:

(18) (SLIDE - THE RECOGNITION PROCESS)

THUS THERE IS A NEED FOR LEVELS OF IDENTIFICATION AND FAMILIARITY, AS WELL AS
FOR LEVELS OF PRIVACY. THERE IS A RATHER UNIVERSAL MECHANISM TO ACCOMPLISH
THESE, KNOWN AS "ESCAPE".

(19) (SLIDE - USAGE OF ESCAPE)

IF YOU ACCEPT MY ARGUMENT OF THE SEPARATION OF THE DATA BASE MANAGEMENT SYSTEM
FROM THE OPERATING SYSTEM, EVEN THOUGH THE ULTIMATE BENEFIT IS NOT SO APPARENT
NOW, THEN YOU WILL SEE THAT THERE ARE MANY THINGS WRONG WITH OUR EXISTING STANDARDS.
FOR EXAMPLE, THE DATA DIVISION OF COBOL IS A PART OF THE PROGRAM, NOT OF THE DATA
TAPE OR OTHER MEDIUM. DESTROY THE PROGRAM AND WHAT IS ON THE TAPE? FURTHERMORE,
THE DATA PROCEDURES ARE NOT COMMON BETWEEN COBOL, PL/I AND FORTRAN. THERE IS NO
REASON THAT THEY SHOULDN'T BE COMMON, AND THE USERS ARE PAYING FOR THIS IN
OPERATING INEFFICIENCY AND UNNECESSARY SOFTWARE USING UP VALUABLE STORAGE.

I DO NOT WISH TO EMPHASIZE STANDARDS OF COMPLIANCE MORE THAN STANDARDS OF
PERFORMANCE. BOTH CONTRIBUTE HEAVILY TO THE EFFICIENCY AND COST-PERFORMANCE
EFFECTIVENESS OF COMPUTER UTILIZATION. IN BOTH AREAS, HOWEVER, I AM AT A GREAT
DISADVANTAGE TO CONVINCING YOU OF THE RELATIVE VALUE OF STANDARDS, AND TO ASK YOUR

SUPPORT IN THEIR CREATION AND ADOPTION. WHY? - BECAUSE MANY OF YOU CANNOT RELATE TO WHAT I AM SAYING, LACKING QUANTITATIVE TOOLS TO MEASURE THE COST OF LACK OR MISUSE OF STANDARDS TO PERFORMANCE.

IF SOME ACTION TAKES 10 MILLISECONDS THAT SHOULD TAKE ONLY ONE, THE HUMAN CANNOT DETECT IT IN HIS SOFTWARE SYSTEM, NOR CAN HE RELATE TO IT WITHOUT MEASUREMENT. WHEN WE INSTRUMENTED THE 600 SOFTWARE (A FIRST IN THE INDUSTRY) WE FOUND SOME PRETTY GHASTLY GLITCHES. CORRECTION HAS ENABLED THE IMPROVEMENT OF PERFORMANCE BY BETTER THAN TWO-TO-ONE. SOME FIRMS NOW SUPPLY INSTRUMENTATION FOR YOUR OWN PROGRAMS, AND HAVE DEMONSTRATED 20 TO 40% PERFORMANCE IMPROVEMENT IN A SHORT TEST TIME. BUT THESE ARE PRIMARILY FOR YOUR OWN PROGRAMS, NOT THE MANUFACTURER- OR SOFTWARE HOUSE-SUPPLIED BASIC SOFTWARE. THE OPERATION OF THIS ^{SOFTWARE IS} PRETTY MUCH OUT OF YOUR CONTROL, AND VERY LIKELY YOU ARE PAYING HEAVILY (AND UNWITTINGLY) FOR TWO THINGS:

- o USELESS PROCESSES

- o INEFFICIENT PROCESSES

STANDARDS AFFECT THE INEFFICIENT PROCESSES, SUCH AS CONVERSION TO AND FROM ASCII CODE IN COMMUNICATION-BASED SYSTEMS. ASK YOUR FRIENDLY COMPUTER SALESMAN TO GUARANTEE TO YOU WHICH ELEMENTS OF THE SOFTWARE SYSTEM ARE THERE TO GET AROUND NON-STANDARDIZATION. ADD UP THE COST, AND YOU AND I CAN TALK ON A MORE UNDERSTANDING BASIS.

I AM NOT INTERESTED IN TAKING UP TIME, OR FILLING UP PAPER, WITH THE TEDIOUS DETAILS OF THE ACTUAL STANDARDIZATION ACTIVITIES. THERE ARE TWO EXCELLENT SOURCES OF SUCH INFORMATION:

1. THE SERIES OF NOTES ON FEDERAL INFORMATION PROCESSING STANDARDS, FROM THE CENTER FOR COMPUTER SCIENCES AND TECHNOLOGY, THE NATIONAL BUREAU OF STANDARDS. THESE ARE AVAILABLE IN THE NBS TECHNICAL NEWS BULLETIN FROM THE SUPT. OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, D.C. A YEARLY SUBSCRIPTION COSTS \$3, AND EVERY COMPUTING INSTALLATION WILL FIND IT WELL WORTH WHILE TO HAVE THESE DOCUMENTS.
2. BEMA, THE BUSINESS EQUIPMENT MANUFACTURERS ASSOCIATION, PUTS OUT A QUARTERLY PROGRESS REPORT ON NATIONAL AND INTERNATIONAL STANDARDIZATION FOR COMPUTERS AND INFORMATION PROCESSING. AVAILABLE UPON REQUEST FROM BEMA, 1828 L STREET NW, WASHINGTON, D.C. 20036.

IT TAKES SOME SACRIFICE TO FOLLOW STANDARDS, AND MORE TO PARTICIPATE IN THEIR DEVELOPMENT. PRESENTLY THE USER FINDS IT DIFFICULT TO SPEND THE EFFORT AND MONEY TO DO SO. NEVERTHELESS, THESE SACRIFICES WILL HAVE TO BE MADE TO ACHIEVE BETTER RESULTS. YOU CAN IMAGINE THE EFFECT UPON THE FRENCH, WITH THEIR NATIONAL AND LINGUISTIC PRIDE, TO WRITE AND DOCUMENT SOFTWARE IN ENGLISH. WE ARE FORTUNATE NOT TO HAVE THAT PROBLEM, SO LET'S DO OUR PART IN OTHER WAYS. IT PAYS OFF.

A SWEDISH FRIEND SAYS THERE IS A STANDARD ANSWER TO THE FEAR THAT A
STANDARDIZED WORLD MIGHT BE AWFULLY DULL. IT IS THAT "A STANDARD-SIZED BRICK
DOESN'T MAKE FOR DULL ARCHITECTURE, AND DON'T FORGET WHAT MOZART DID WITH ALL
THOSE STANDARDIZED LITTLE NOTES." WE'RE GOING TO BE IN AN AWFUL MUDDLE AS
THE LARGEST BUSINESS IN THE POST-INDUSTRIAL WORLD IF WE CAN'T BRING SOME
ORDER INTO ~~IT~~ THROUGH STANDARDS. WE ARE OVERDUE IN STARTING AN EFFORT OF THE
REQUIRED MAGNITUDE.

ACM 70 has many facets but the central point is an attempt to determine the users' needs for the next decade

What Is ACM 70?

by R. W. Bemer

G The ACM 70 Conference could have been the 25th annual ritual of a society spawned and nurtured by the electronic computer, dominated by technocentric interests, a little bit insolvent, and not caring how its machine influenced society—just as long as they were free to have fun with it, parse yet another programming language, argue about computational accuracy with the fervor of an early tabulator of angels on the head of a pin, and maintain a lovely insularity from people who did not talk their own jargon. This will not be so, however, thanks to the mass awakening to a new set of values that most people are experiencing.

Without meaning to downgrade the value of efforts 1 through 24, let us see what effort 25 is about. Surely the main characteristic is that it is about many things, and they are:

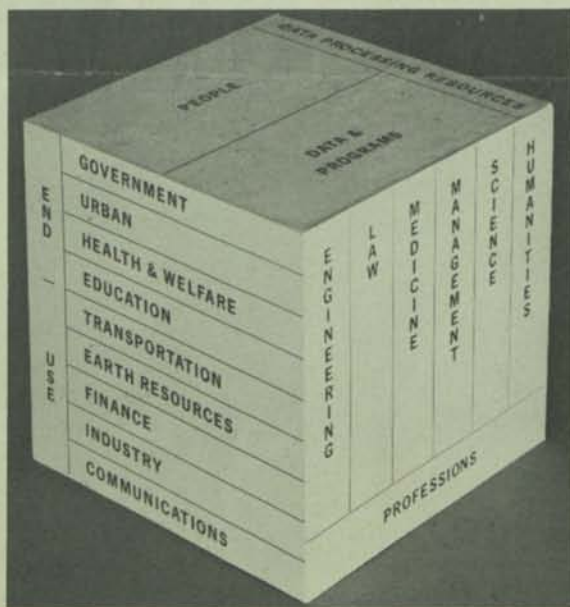
It's a model of an activity. The structure of ACM 70 is shown in the figure at the right.

The activity is to enumerate the information processing requirements of the several end-users and professions for the next decade, and plan the optimum way to allocate resources and development in order to meet those requirements. Note that the intersections of sectors from each face are usually meaningful in some degree (i.e., *management data in transportation*). Overlaps and even conflicts can exist, but these have been worked out cooperatively.

This activity can be ACM 70. It can also be ACM 71 and 72. It can be a national computer year on the lines of the International Geophysical Year. It can be an international computer year activity. Each could

grow from the other with the addition of time, effort, people and scope.

It's a conference. ACM 70 takes place Sept. 1-3, at the New York Hilton Hotel. It will undoubtedly be attended by many members of ACM. However, unless the planners are all dreaming, it will be attended by a host of people who may never have heard of the ACM, but are nevertheless very much impacted by computers in their work. Special one-day registrations have



been provided for the convenience of working people—doctors, lawyers, Wall Street men, city planners, etc.

The chart below shows the conference schedule.

It's a plan. If we are to make best use of computer systems in the next decade we will have to have a good plan. Coming before any plan are the goals to be achieved. The ACM 70 goals are:

1. To consciously put computers in service to national goals, to increase public understanding of the role and potential of computer usage, and to accent the role of the computer as servant.

2. To develop strategies for the best future use of computer systems (technological, social, educational, political, legislative).

3. To conserve, and maximize utility of, those existing and future intellectual resources known as data

and programs by finding how to utilize them on multiple equipments and in multiple applications.

4. To aid government, business and private decisions by opening up new and more complete data for those decisions, and to facilitate making of these decisions by reducing the information volume required (as opposed to data volume).

5. To plan a closed cycle for redistributing work assignments between people and computers, for re-education prior to change, so that our citizens can best fulfill their potential.

6. To ensure that public safety and welfare are considered adequately when computers are integrated directly into human activity.

7. To set up new and broad interdisciplinary information exchange paths among hitherto segregated organizations, and to foster their maximum involve-

CONFERENCE AT A GLANCE

SECTORS	PAGE	TUESDAY (SEPT. 1)				WEDNESDAY (SEPT. 2)				THURSDAY (SEPT. 3)	
		9:00 - 11:00	11:00 - 12:30	2:00 - 6:00	EVENG.	9:00 - 12:00	12:00 - 2:00	2:00 - 6:00	EVENG.	9:00 - 12:30	2:00 - 5:00
Communications	13	A Communication Aspects (1)				B Communication Aspects (2)					
Data & Programs	14	A Program Classification *									
Earth Resources	14		A User Requirements			B Sensor Acquisition		C Data Systems Requirements			D Earth Resource Management
Education	18	A Computers in College & Library	B Computers for Instructions	C System Analysis		D Programmer & Operator Train.		E Comp. Science Education			F Computers & Cont'd. Education
Engineering	25		A Management & Design Eng's			B Computer Graphics in 70's		C Design Eng's 2:00 - 4:00 p.m.			D Production Engineering
Finance	29					A Securities & Insurance		B Banking & Accounting			
Government	34		A Computers in Government								
Health & Welfare	35							A Computers in Health, Welfare			
Humanities	36	A Computerization in Liberal Arts									
Industry	38					A Computers in Industry (1)		B Computers in Industry (2)			C Computers in Industry (3)
Law	40	**									A National Crime Information Ctr.
Management	41		A Administration & Education								B Information Syst. & Future Mgmt.
Medicine	44					Computers & Practice of Med. Medical Research					
People	46	A Select. Computer Work Force	B Human Factors 4:30 - 6:00 p.m.					C Impact on the Public			
Science	48							A Scientific Appl. 4:00 - 6:00 p.m.			B Scientific Computer Syst.
Transportation	51		A Computer Transp. 4:00 - 6:00 p.m.								
Urban	52	A Computers in Future Cities	B Urban Developmt. 2:00 - 4:00 p.m.								

* This session has been rescheduled to 9:00 to 11:00, Thurs. Sept. 3.
 ** Legal Assistance via Computer session has been scheduled for this time.

What is ACM 70?...

ment on a national scale.

8. To plan the most economical and effective interaction between computing and other systems, such as communications.

Undoubtedly it will take more than ACM 70 to meet these goals, but this is the start.

It's finding where we are. Only the newest or most myopic participants will believe that the computer business knows where it is and is going. Government estimates of the number of computers expected to be in use by 1975 vary from 100,000 to 330,000 (just actual count, no mention of processing power—which can vary by factors of 50 without too much difficulty). Private census can barely enumerate existing usage, much less predict the proportion five years from now. Industries with slow growth rates can perhaps afford to react; with high growth rates we must plan ahead on a nationwide scale.

How are computers used?

On Dec. 20, 1968, the U.N. General Assembly asked the Secretary General to find out how computers were being used in the various countries, in order to apply them with maximum utility to the economic and social development of all peoples.

Accordingly the member countries were asked what amounted to "How do you use computers?" and to kindly reply by Oct. 15, 1969. Many countries did reply. Some of the most comprehensive returns were from the U.K., Japan, and Israel. Unfortunately missing was a response from the world's greatest and probably most knowledgeable user of computers. We may conjecture that either the State Department did not know how to contact a \$14 billion yearly industry, or that the huge industry itself did not really know how those computers were used, at least in the sense of being able to make a coherent reply.

It's an affirmation of responsibility. To be repetitive, it is publicly obvious that much of the world's population has been shirking its social responsibilities consistently. Science is now under suspicion as being a definable subset of the shirkers. To our horror, the sub-subset of computer science has found that its baby is the handy focus of much antagonism. Call any business to complain of an error and the clerk is likely to answer: "Sorry. We have a computer now, and it doesn't work right." The answer is never that the computer works but that the humans who programmed it were at fault, or that the computer was integrated into human activities without sufficient attention to safeguards. Probably the sociologists and psychologists can explain the reaction in technical terms, but explaining does not solve the problem.

As the time rolled around to plan yet another conference, it was painful for the ACM management to face the situation, as it would be for any management. In the end, they bit the bullet, as they must. The road to health must start with inward determination.

It's a new look. Until now computer conferences have been characterizable as either vendors talking to vendors, or users talking to users. The new look is that the users have been asked to give the computer industry their best picture of their total information processing needs for the next decade.

In line with this, there have been no unsolicited or

refereed papers for this conference. Nor will one hear papers on microprogramming, fast adders, or the like. The sector chairmen were selected for their knowledge of the user sector, not familiarity with computers necessarily. Thus they knew the body of competent and qualified people to turn to for an exposition of this nature.

If it should turn out that their needs cannot be enunciated well, then at least we have tried and opened a door. Contrariwise (in Alice in Wonderland terminology, which is very much like our own), the odds are high that there are users just panting for the chance to tell the computer industry—on a total and nationwide basis, mind you—what they believe should be produced.

It's a new direction. At the least, ACM 70 will pose questions that ACM 71 and 72 can try to answer, and give continuing purpose to those activities. Very likely, if the 1970 conference meets enough of its goals, the whole fabric of ACM can be rewoven. Like other professional societies, ACM needs periodic redefinition of its goals, membership growth consistent with industry growth, and means of giving good services to its members. However, ACM also needs to find better ways of serving the public. Carl Frey, executive director of the Engineers Joint Council, reminds us that organizations of this type are tax-exempt under statute 501(c)(3) only if the main thrust of their total activity is in the public interest, and not solely for the benefit of its members.

Mr. Frey also reports that the American Society of Association Executives evidences a growing feeling that there is also a collective responsibility owed society which may be impossible to fulfill by individual efforts of single societies. Much more cooperative effort is required, and ACM 70 is the very model of such a cooperative effort in solving a complex problem.

It's a hope. Surely the case can be made that computers yield more benefit than harm. The hope is that this excess can be increased and maximized.

When it can be demonstrated that computers are for people, then people will be for computers. To achieve this will require a very conscious effort. In the end, that's really what ACM 70 is. ■

(The following articles are a sample of the sector activities planned for ACM 70.)



Mr. Bemer is manager of systems and software engineering integration for General Electric and is program chairman for the ACM's 25th annual convention. He is noted for his contributions to such organizations as IFIP, ANSI, ECMA, and CODASYL and as an author, editor, and developer of computer techniques.

CodasyL 20

AGENDA
CODASYL 20TH ANNIVERSARY
MAY 21-22, 1979
WASHINGTON, DC

MONDAY, MAY 21, 1979

- 0800-0900 Registration - Capitol View Ballroom
- 0900-0915 Welcoming Address
John L. Jones, Vice President, Southern Railway
System, Chairman, Executive Committee
- 0915-0945 Keynote Address
The Honorable Elmer B. Staats
Comptroller General of the United States
- 0945-1000 Coffee Break
- 1000-1045 Presentation of CodasyL System Architecture
Mr. Richard Kurz, Southern Railway System
Executive Committee and Committee Chairman
- 1045-1115 Report of the Cobol Committee
Mr. Donald F. Nelson, Control Data Corporation
Chairman, Cobol Committee
- 1115-1200 Presentation - Distributed System Report
Mr. William H. Stieger, Standard Oil Company (Ohio)
Chairman, Systems Committee
- 1200-1400 Luncheon - "Reminiscing"
Speaker Bob Bemer, Honeywell
- 1400-1430 Data Description Language Committee Report
Mr. Michael L. O'Connell, Digital Equipment
Corporation, Chairman, DDLC
- 1430-1515 CodasyL Data Base Implementation
Mr. John Cullinane
President, Cullinane Corporation

AGENDA
PAGE TWO

MONDAY, MAY 21, 1979 (Con't)

- 1515-1600 TOTAL - Another Approach to Data Base
Mr. Tom Nies, President, CINCOM
- 1600-1615 Coffee Break
- 1615-1700 Case Study of a Distributed Processor System
Using Mini-Computers
Mr. Mayford Roark, Executive Director
Ford Motor Company Systems Office
- 1700-1730 Question and Answer Period
- 1800 Reception (Cash Bar)

TUESDAY, MAY 22, 1979

- 0900-0930 Address by the Honorable William G. Claytor, Jr.
Secretary of the Navy
- 0930-1015 Presentation of EUFC Report
Dr. H. C. Lefkovits, H. C. Lefkovits Associates
Chairman, End User Facilities Committee
- 1015-1030 Coffee Break
- 1030-1115 Presentation of Common Operating System Control
Language, Mr. Thomas Harris
Chairman, Common Operating Systems Control Language
- 1115-1200 Question and Answer Period - All Committee
Chairmen
- 1200 Adjournment

Luncheon Talk,
20th Anniversary of CODASYL
1979 May 21
R. W. Bemer

(Introduction)

The opportunity to talk to you all here has of course triggered a lot of recollections. For example, I remember that the Short Range Committee that gave us the first cut at COBOL was just that. They were supposed to get it on within 3 months. Grace Hopper and I snickered about the impossibly short time, and it appears we were justified. COBOL isn't finished yet, as we celebrate 20 years of work!

Surely you remember COBOL 60. And COBOL 61. And COBOL 68. And COBOL 74. And COBOL 80. And look forward to COBOL 2000. No wonder the inhabitants of Battlestar Galactica worshipped Lord COBOL. How immortal can you get?

COBOL must have been a significant contribution, measured by how the early proponents have been honored. Of course, Charlie Phillips and Joe Cunningham were very distinguished when they moved the project. For some others --

Do any other countries in the world have high-ranking women officers that are so important they are not allowed to retire, like our own Navy Captain Grace Murray Hopper? But I'm not sure what vessels she may board besides the USS Constitution.

And who was the first woman elected president of the Association for Computing Machinery? Jean Sammet.

Would the Honorable Jack Jones have kept a Vice Presidency so long in a large company if he hadn't been so closely associated with CODASYL and COBOL? Well, yes, he would. But is that any way to run a railroad?

And then there are those who prefer being their own boss -- like Howard Bromberg, who took off for San Francisco as soon as he heard about North Beach.

A Flash of History

Due to the foresight of the founders, and their cleverness in keeping the history in print via hundreds of thousands of copies of the COBOL specification, I need go into very little history here. We are, of course, celebrating that first meeting at the Pentagon, on 1959 May 28 and 29.

It was a noble venture. Jean Sammet's history quotes a motive as articulated by Charlie Phillips -- "To broaden the base of those who can state problems to computers". Possibly he actually said it; I have a great admiration for Charlie. Particularly when you consider that he has been active in CODASYL eight years longer than the formal reign of the Shah of Iran! And with every prospect of running up the total!

Jean also allowed as how the Short Range Committee was somewhat deluded, thinking that their first spec was not "something intended for longevity". But longevity it has, of course, due to the renewability permitted by the CODASYL framework. Which is the principle of my home town -- Phoenix. It would be interesting to compare today's grown-up COBOL with that upstart FACT language of Honeywell, that caused such a stir then!

The name of Jean Sammet also reminds me of a time when my wife Marion and I were both working at IBM. Seems that I spoke of Jean Sammet and COBOL quite often, to the point where my wife became so curious that it could change to jealous. But it blew over when I explained that COBOL wasn't a perfume!

Passing quickly over history, you recall that the Short Range Committee did comply very well, and their report was accepted on 1960 January 7. This led to submission for printing in April, and actual publication via the US GPO in June.

Of course all these activities caught the public eye. Business Week had been on top of the situation since June of 1959, and in April of 1960 the effort was exposed in Computing News, Issue 171. The insight of its editor, Jackson Granholm, was so penetrating that I would like to recall it to you (mainly because the New York Public Library has no back issues -- it's rumored that they were burned):

In a masterful piece of reporting, entitled "POOBLE-ORIENTED LANGUAGES", we find -

"That the eminent Dr. Rupert B. Pooble should concern himself with the subject of programming languages was to be expected. After all, in his position as Director of Mathematical Action for the Inscrutable Atomic Corporation, Pooble swung a big mass ...

Therefore, as fate would have it, it was on a well-known Tuesday during November past that Pooble called a meeting in his large, oak-paneled office. To this meeting he rather arbitrarily summoned practically everyone in the manufacturing end of the industry who could see lightning and hear thunder.

It was apparent early in the gathering that the attendees tended to break pretty well into two camps. These two camps, to coin some cliches, might well be described as the "Know-Nothings" and the "Green-back Party".

The Know-Nothings lined up solidly behind their idol, Horton Dreamer, Associate Director of Programmercraft for the Suffix-Specific Division of Quantum-Occluded-Domineer. The Green-back party, on the other hand, were solidly behind their eminent spokesman, Dr. Mary Margaret Groper of the Competing Equipment Corporation of America.

Pooble was quick to get to the point.

"It is manifest", he said, in his resonant, cultured voice, "that Inscrutable Atomic is the biggest computing machine customer in the world. We have at this very moment in the back room a total of 43 electronic computers of various sizes. These machines are on rental from 17 different manufacturers ... However", Pooble continued, "I am sorry to note that these 43 machines are programmed in no less than 678 systems of pseudocoding, not to mention their own unique machine codes".

"My!" said Dr. Groper.

"I have asked you here to see what you intend to do about it", Pooble said.

"Not a damn thing", said Dreamer, "If you'd stuck with our equipment like any sensible person you wouldn't be in this mess".

"Now just a dog-boned minute, Horton", Pooble said, his face growing crimson, "if you don't want your rent cut off you better shape up better than that".

"No need to get hot under the collar, Rupert", said Dreamer, "but they warned me at headquarters that you were apt to pull some glitch like this".

"Never mind that", Pooble said. "What I expect you people to do is to form a committee to produce, at no cost to Inscrutable, the ultimate programming language".

"You're daft", Dr. Groper observed.

"Nonetheless, you have two weeks to get started, or we go back to desk calculators".

Manifestly the heat was on, and the attendees, knowing on which side their bread was peanut-buttered, got with it with dispatch.

By 2:30 in the afternoon a name had been selected. It was decided that the ultimate pseudocode would be called "POOGOL", for Popular Operational Ordinary Glitch-Oriented Language.

By the time the meeting broke up, a modest little working group of 310 members had been set up to implement POOGOL in three months (sic). Dr. Groper and Horton Dreamer were appointed co-chairmen. Since they didn't speak to each other this made for rather difficult coordination, but at least the job was under way.

There's more, the story going that the project lost it's driving force. Pooble resigned and went to work for Dreamer at twice the salary. Etc.

There may have been fallout from this reportage. When Charlie Phillips was first at BEMA, it was in New York, whose citizens drop some R's. A number of people had the misapprehension that he worked for me. This problem has now been rectified by adding a "C", to get CBEMA, whereas I'm RBEMA.

And one wonders whether Bromberg read the story, having it still on his mind as he passed a certain stoneworks holding a sale on animal tombstones!

Original Success of CODASYL

Although the activities are now more varied, the original success of CODASYL was the COBOL language. And it's worth reminding ourselves why that should have been so.

It wasn't that it offered a substantial set of capabilities not previously available (B.C.). The official history of COBOL recounts the proprietary languages that were absorbed, melted, and recast into COBOL. The features of each could be found, in a form that, when altered, usually was free of the prejudice and provincialism of the originator (who probably found it somewhere else, anyway).

It wasn't that the computer system suppliers of the time, overcome by user-inspired altruism and conviction that their customers knew more than they did, decided to open the halls for a camel-building party. So they could laugh at it later and hawk their own consistent product, not made by committee. Even if so, the Short-Range Committee fooled them by eventually releasing a remarkably wellmade specification, quite unlike a camel. But then when we had it, not all suppliers rushed to make COBOL their product. In at least one case, it took much arm-twisting by users, and prospective users, to get COBOL compilers in the catalog. COMTRAN isn't much remembered as one of the acronyms I devised. Fortunately CODASYL is.

Incidentally, I usually pronounce it like "codicil" in a will, because that's how it came to mind, rather than like a musical "coda". But this is one place I don't worry about agreeing upon a standard.

COBOL was successful because of the "CO" in its name. COMMON. Not common in the vulgar sense, but common because the programs using it were fairly portable to other computers. With business data that's a lot tougher job than with floating point numbers!

COBOL was the seventh language to work on more than one model of computer, (Fortran, Algol, APT, IT, Mystic, IPL) and the third (APT, Algol) to work on models of more than one manufacturer. But it was the first business language on both counts, proved on December 6 of 1960.

Unique Role of CODASYL

The significance of CODASYL is greater than ever now. Our economic struggle is no longer within our own country. It's with other countries. And we're not doing so well, or haven't you noticed? Computers and electronic gear account for a substantial plus in our balance of payments act. The trade figures for March were released last week. The total US deficit was \$821 million. Computer exports were \$496 million, imports \$68 million, for a net surplus of \$428 million. Without computers as a viable business, then, the US deficit would have been 52% worse! So we should be careful to maintain this advantage. Do you think we can?

Last year a Frenchman told me about a business dinner he had attended in Japan. The Japanese executive next to him had become convivial enough to say:

"Do you remember the German cameras?

All Japanese now ...

Remember the Swiss watches?

All Japanese now ...

Remember the American computers? ..."

If you're thinking "What does this have to do with CODASYL?", let me remind you of this country's antitrust laws. They may have had ample justification when passed. In each company I have worked for, I have been instructed firmly in the limits of my participation in standards activities of all kinds. Do nothing to contravene the antitrust laws, they say.

But the Japanese are not hampered that way in cooperative ventures. Just the opposite. They have government-controlled joint research, shared between companies. Free enterprise and traditional American competition are penalized in that game. Antitrust limits such cooperation between companies (although last week the government permitted Chrysler to buy some research results from GM, so maybe they're wising up).

So who can help the U. S. to keep a competitive edge via cooperative development efforts?

- o Not ANSI. Their charter is to register standards, not develop or legislate them. They cater to more fields than just computers.
- o Professional societies can attract volunteers, but the work would have to be funded by dues. These twenty years have shown how inadequately that works. Example: ACM took 6 months after CODASYL started to even acknowledge the work.
- o The several user groups can't be expected to maintain the necessary broad scope and viewpoint required for full portability.
- o ECMA (European Computer Manufacturers Association) has done excellent work, but we can't entrust our export balancing act to them.

Strength of a country, like corporate strength, can come from cooperative ventures. From combined R&D.

Let us be grateful that CODASYL exists today, for we might not be able to establish it new today. Reminds me of a T-shirt slogan - "Do it now, before it becomes illegal". CODASYL is unique. Its fairness and propriety have been established.

It's a DATA World

It's heartening that today's meeting confirms that we backed the right horse. It's Committee on DATA Systems Languages. Most computer usage was still for numerical calculation then; at least the visible emphasis was. Papers on data manipulation were few at the conferences of the day.

I said then that business problems were 10-20 times as difficult as numerical problems, which was not a popular opinion. But it hasn't been until the last few years, working with live databases, that I realize -- in simple enough terms -- why this is so.

Computers can usually process an input number, for it may lie anywhere within a known spectrum according to certain formation rules. Non-numeric data won't work that way. More representations are possible, which is why we have alphabetic license plates, and why the Post Office wants to add four more digits to the ZIPcode. And they'll probably still insist that you give the city and state, too, or they won't send the mail.

That's the key to data processing -- pattern matching. A pattern of bits, of characters, of words, or of total behavior. It's still the key.

Real World Databases

Two years ago I was asked to demonstrate some relational database processing to a certain government agency. I was to leave Phoenix on Sunday, and their sample data just arrived Friday afternoon. I went home, fired up the terminal, built a Martini, and looked at the file which had been loaded. It appeared to be all capitals, and studded with spaces, like maybe they had used an old-fashioned keypunch to enter the data. So I used TEX to replace all double spaces with a separator, until none were left. Storing the file under a new name showed that it only took half as much space in that form! So they were paying their supplier twice too much for disk packs and drives.

But that's not the most important part. The concordance I ran was astonishing. It said that a certain petroleum company had three high-level executives with sound-alike names -- Wohlegemuth, Wohlgemuth, and Wolgemuth.

You've guessed that they are really just one man, that the database was dirty. Never mind, on a query they're only going to find him one time out of three. Do you want our government to make decisions from such data? Or do you think that maybe that's what the problem is?

That's what I mean by realworld databases and realworld people associated with entry and processing of the data. Note that CODASYL DBTG schemas would also fail in this instance if there were pointers to all three men. In the real world one has fuzzy sets. "His name sounds like ... ", or "I think it began with Gy". And in the real world one doesn't always know in advance what is expected to be extracted from a database upon query or display of some subset. One doesn't put in pointers to all of the other people in the entire world that have 1973 blue Dodges. Just as playing chess by computer cannot be done by projecting ahead all possible results of the next nine moves. It gets too astronomical.

Surely we see from the present usage of micro-computers that many of our previous tasks will be transferred to them. What's left for large computers to do? Manipulate databases, for one. Take in, via communications, the small private databases, agglomerate them, and parse them to extract new information that was previously unsuspected or unavailable to the single owner.

I don't mean to downgrade the DBTG work (and get shot by my colleague Bachman). It's a most important tool, but not the last that CODASYL should concern itself with in the database world. Parsing with pattern-matching devices of sufficient ingenuity can handle those "fuzzy sets". And it can handle the huge number of today's existing databases that are not of the pointered type.

Brief Recommendations

Peter Landin's paper "The Next 700 Programming Languages" said that if a user body becomes large enough it is economically viable to specialize to segments of usage. We must remember that we must supply tools for the pre-DBTG users, and for the post-DBTG users, as exemplified by the microcomputer and communication networks.

With this in mind I offer CODASYL some brief and modest recommendations on future actions and needs. They may be covered fully already. If so, ignore my points as fulfilled. Here they are:

1. In your database work, accord equal rights to all programming languages. Right now, COBOL is favored like a sprained ankle over FORTRAN.
2. Give consideration to pointerless databases. Consider flat files and co-files for them. Memory costs say we can do it now, and make a database understandable by simply printing it serially. Support efforts to reserve the upper half of ASCII for tokens, not printing characters (which are adequately covered by code extension). Then you may have flat files describing the token-to-actual relationships, but run the relational database with tokens only.
3. Continue the good work for the end-users, but remember that we may wish to manipulate a database, not just interrogate it.
4. Continue with the operating system command language work, but remember that it can all be done with a text processing language. Witness the success of the UNIX system and TEX. Keep close liaison between the COSCL work and the "Nicola" project under W. German government sponsorship. Mapping COSCL to Nicola will be your test of success.

5. Gerry Weinberg says "While 80% of commercial applications programming is done in COBOL, not even 5% of the programming literature deals with COBOL". What are you going to do about it?
6. Carry the COBOL lesson of levels and modules one step further, Carry it to many languages, not one. What's so particularly "database" about COBOL? You're answering that with FORTRAN, at least. But not with all of the important languages -- including BASIC, PASCAL, and PL/I. What's so particularly "realtime" about PL/I, that doesn't apply to COBOL, FORTRAN, CORAL, PEARL, etc.?

There's a way to get rid of expensive duplication between so many programming languages. CODASYL could sponsor work to extract the common parts of all of these languages. Then the standard specification for PASCAL could say "See CODASYL standard referent 6.2.3 for this function".

Look at any big operating system. It's compiling source programs in several programming languages simultaneously, and each compiler has its own code for the functions otherwise identical in each language. I don't worry about saving memory. It's the building and maintenance of duplicate and redundant software modules that is bad. Landin says to cloak the function in whatever language is easiest for your audience and users. But I say -- don't use that as an excuse to build duplicate software.

Conclusion

The success of COBOL has been overwhelming. Its aspects and concepts have entered and modified our way of life in many ways. The 1979 March 15 issue of Computing (the British weekly) reports hearing a luncheon conversation like this: "As a manager, he's all Data Division and no Procedure Division".

Nevertheless, I want to remind you that the job isn't done, it can be done better, and it will pay off to do it better. Get in step with the micro-computer people; they're doing things we have said couldn't be done.

It has been fun talking to you. I hope that we have new achievements to be proud of ten years from now. I want to express, on your behalf and mine, appreciation for the vision of the known pioneers, for those who have left (like Roy Goldfinger), for those that worked behind the scenes (like Mary Hawes, Saul Gorn, Walter Carlson, and many others), for Rik Blasius and the helpful Canadian Government, and for those who have labored these two decades to enlarge and improve the work.