Programming Languages: History and Future

Jean E. Sammet
IBM Corporation

This paper discusses both the history and future of programming languages (= higher level languages). Some of the difficulties in writing such a history are indicated. A key part of the paper is a tree showing the chronological development of languages and their interrelationships. Reasons for the proliferation of languages are given. The major languages are listed with the reasons for their importance. A section on chronology indicates the happenings of the significant previous time periods and the major topics of 1972. Key concepts other than specific languages are discussed.

Key Words and Phrases: programming languages, higher level languages, languages, history, future directions, language interrelationships, programming language tree, programming language history, programming language future

CR Categories: 1.2, 4.2

 This paper, which includes the Language History Chart, in substance will form part of the 2nd edition of the book, *Programming Languages: History and Fundamentals* by Jean E. Sammet, which is currently in preparation and is expected to be published by Prentice-Hall, Inc.

Author's address: IBM Corporation, 545 Technology Square, Cambridge, MA 02139. All comments and opinions in this paper represent (only) the personal views of the author.

[1] References have been kept to a minimum to save space. Any language for which no reference is given in this paper has the citations in either Sammet [14] or [16]; almost all are in the former.

1. Introduction

1.1 Definition of Programming Languages

It is well known by now that there is no general agreement on the meaning of the term "programming languages." In this paper the term is used interchangeably with "higher level languages," but even that does not solve the definition problem. Hence it is necessary to specify that a programming language is considered to be a set of characters and rules for combining them which have the following characteristics: (1) machine code knowledge is unnecessary; (2) there is good potential for conversion to other computers; (3) there is an instruction explosion (from one to many); and (4) there is a notation which is closer to the original problem than assembly language would be.

1.2 Purpose and Scope of Paper

Programming languages are almost as old as ACM, since the latter started in 1947 while the former started in 1952 with Short Code for UNIVAC. Since then, some efforts—although relatively few—have been made to record certain historical aspects of this field, e.g. Rosen [12, 13], Bemer [6], Sammet [14].[1] Every author, including this one, takes a different point of view of what is important, how to define and interpret dates, etc. In this paper I hope to indicate some of the problems in actually writing about the history of programming languages, and then make my own attempt to provide perspective. This latter seems extremely important because there are over 170 languages in use just in the United States in 1972. While about half of these fall into the category of "languages for specialized application areas" and are discussed in more detail elsewhere [18], another 85 still remain to be accounted for in some way. What is it about this field that causes such proliferation? Is there any sense to what has happened?

What—if any—are the interrelationships among the hundreds of languages which have been developed since 1952? Finally, and perhaps most important, what does this portend? Answers to these questions must be limited because of space and time constraints.

There are three facets of programming language history which this paper does *not* discuss. One is the development of any individual language; this has been done in considerable detail by the author [14]. A second is the set of specific concepts in individual languages. The third omission is any discussion of implementation.

1.3 Problems in Discussing History of Programming Languages

There are several problems in writing about the history of programming languages. For example, in considering programming languages there are a number of phases, each of which is important in the overall development, but which is almost impossible *in retrospect* to pinpoint to an exact time. We—as a profession and an industry—tend to toss around dates without being very specific about what they mean. Thus various people would define "the earliest date" for COBOL as 1959, 1960, or 1961, and yet COBOL has one of the clearest and best-documented histories. In order to define the problem of establishing dates, note that at least the following phases exist for programming languages (and in fact most of these apply to any program). Some phases may occur in parallel.

1. Initial idea.
2. First documentation of initial idea.
3. Preliminary specifications.
4. "Final" specifications (i.e. those which will be implemented).
5. "Prototype" running (i.e. as thoroughly debugged as the state of the art will permit, but perhaps not all of the features are included).
6. "Full language" compiler (or interpreter) is running.
7. Usage on real problems by the developers.
8. Users (on real problems) other than the developers.
9. Documentation, e. g. manuals and tutorials.
10. Paper in professional journal or conference. (Sometimes this appears as early as step 3.)
11. Extension, modification, or new version.

Note that a formal definition might appear at steps 3, 4, and 9. Also note that some of these steps may occur simultaneously or be so combined that they are indistinguishable. Furthermore, step 11 allows recycling to any lower number and repeating the cycle for a new version. *In only a few instances are dates easily available for each of these phases.*

Another aspect of the problem in writing a history of programming languages pertains to the definition of a language and variations of it. This makes establishing accurate counts impossible. Even if we assume that we can recognize a higher level language when we see one, we don't have a rigorous (or even weak) way of knowing whether we merely have a dialect or whether we really have a new language. This point is discussed by the author in much greater length in [17]. As one example among many languages with similar situations, note the existence of SIMSCRIPT I, I.5, II, II.5, and II Plus. It is clear that these languages are related and yet different. Are they five separate languages, or only one?

2. Historical Development of Languages

This section shows the historical development and interrelationship of languages by means of a chart. Reasons for the proliferation of languages and for the importance of languages are given. Finally, there is a listing—with justification—of the most important languages.

2.1 Development by Dates and Interrelationships

There are three major ways to describe the historical development of programming languages: first, events for each language, second, events within a given year, and third, the relationships and descendants of languages. As previously stated, the first has already been done by this author, so only the second and third are needed here to provide a (hopefully) solid foundation of facts. The chart covers the second and third ways by providing a tree depicting yearly information at a glance, but it also shows the influence and the descendants of the languages. Such a chart is of course subject to space and graphic constraints; an enlargement of almost any portion would provide useful information.

Only separate version numbers for fairly significant languages (e.g. FORTRAN) have been shown. The legend on the chart indicates that the reader can determine the earliest date known to the author pertaining to a given language or, alternatively, the particular phase of development (see Section 1.3) for which information is clearly identified. Languages for specialized application areas (e.g. simulation, machine tool control, civil engineering) are not included on the chart because the number of them would have doubled the complexity.

The lists of languages shown on the back of the chart represent snapshots at three points in time. Only ten languages, namely ALGOL 60, APT, COBOL, COMIT, FORTRAN, IPL-V, LISP 1.5, MAD, MADCAP, and NELIAC appear on all three lists and thus indicate continuous usage.

2.2 Reasons for Language Proliferation

The incredible proliferation of higher level languages appears to be caused by one or more of the following:

1. A really new language concept has been developed and/or a new application area is deemed worthy of having its own language.
2. After experience with a particular language, its deficiencies are clear enough that a complete new language is needed—and justifiably created—to correct them.
3. Facilities of several languages are best combined into a single new language.

4. It is felt to be easier to get additional capability or changes in style with a new language rather than to extend or modify an existing one.
5. It is fun to design and implement a new language, and someone wants to do it and can obtain the funds.
6. There are personal preferences and prejudices against the existing languages even though one of these languages might serve the purpose for which the new language is intended.
7. The developer is unaware of the existence of a language that meets his needs, so he creates his own while believing he is meeting the conditions of (1) or (2).

There are only a few illustrations of (1) and (2). Examples of (1) include—but are not limited to—FORTRAN, APT, FLOWMATIC, IPL, GPSS, COGO. Major examples of (2) are COBOL (to replace FLOWMATIC)and SNOBOL (to replace COMIT).

2.3 Reasons for Importance of a Language

What are the reasons that one language becomes widely used or considered significant or both, while others remain for all practical purposes the property of a small group? The generally obvious answer is practicality; i.e. the language is suitable for a significant (although not necessarily large) class of problems and good compilers can be written for it. However, those are only the most obvious attributes, and underneath them lie a number of other factors, not all of them based on facts. For example, the psychological issue of snob appeal is more important than many people might think. Thus, the personal prestige and leadership (or lack thereof) of those individuals who are involved in the development of a language play an enormous role.

Some languages clearly create a spark, which causes the languages to become popular—sometimes to a level of fanaticism—regardless of the difficulties. This is equivalent to the "political charisma" which often affects election results. The best example of languages which inspire some people are ALGOL and APL\360; on the other hand neither BASIC (at the small end) nor PL/I (at the large end) generates as much *personal* enthusiasm by the users. It is hard to pinpoint the reasons for lack of charisma in a language, and it has very little to do with the actual use. For example, BASIC and COBOL are very widely used languages, but I doubt whether many people are personally enthusiastic about either of them.

In summary, there are really two major reasons for a language to be considered significant: one is that it is economically practical and hence very useful, and the other is that it is technically new. In the next section, I have used both criteria. There doesn't seem to be any language which met both at its inception; this is not surprising.

2.4 Major Languages

In one sense, the items on a list of the major programming languages are obvious to almost everyone; however, in reality it is unlikely that any two people would agree on what the list should contain! Speaking purely as an individual, and with due concern for my prejudices, I wish to indicate the languages which I consider of major significance, along with the reasons for their importance.

Note that importance and wide use are not the same; several of the languages on the list have been used only by a relatively small number of people, or they have been implemented on only one computer.

In approximate chronological order the languages of major significance are:

APT. The first language for a specialized application area.

FORTRAN. The first higher level language to be widely used. It opened the door to practical usage of computers by large numbers of scientific and engineering personnel.

FLOWMATIC. The first language suitable for business data processing and the first to have heavy emphasis on an "English-like" syntax.

IPL-V. The first—and also a major—language for doing list processing.

COMIT. The first realistic string handling and pattern matching language; most of its features appear (although with different syntax) in any other language attempting to do any string manipulation.

COBOL. One of the most widely used languages on an absolute basis, and the most widely used for business applications. Technical attributes include real attempts at an English-like syntax and at machine independence.

ALGOL 60. Introduced many specific features in an elegant fashion, and combined with its formal syntactic definition, inspired most of the theoretical work in programming languages and much of the work on implementation techniques. More widely used in Europe than in the United States.

LISP. Introduced concepts of functional programming combined with facility for doing list processing. Used by many of the people working in the field of artificial intelligence.

JOVIAL. The first language to include adequate capability for handling scientific computations, input/output, logical manipulation of information, and data storage and handling. Most JOVIAL compilers were written in JOVIAL.

GPSS. The first language which made simulation a practical tool for most people.

JOSS. The first interactive language; it spawned a number of dialects which collectively helped to make time-sharing practical for computational problems.

FORMAC. The first language to be used fairly widely on a practical basis for mathematical problems needing formula manipulation.

APL\360. Provided many higher level operators, which permitted extremely short algorithms and caused new ways of looking at some problems.

The prime characteristic of this list of major languages is that each language is unique in some way; put another way, each language is independent of any other on the list (except COBOL which drew heavily on FLOW-MATIC experience). Furthermore, each was apparently developed independently of any other, although each developing group was (presumably) aware of the other languages existing at the time of the development. Some other languages are now more widely used or more comprehensive than those on the list, specifically BASIC, PL/I, SIMSCRIPT, and SNOBOL. In many cases, they have almost completely replaced some of the languages on the list (e.g. BASIC for JOSS and its derivatives, SNOBOL for COMIT). The four "obvious candidates" cited above are omitted for the following reasons: BASIC, although simple and economical, added no new concepts, was not the first on-line language, and was not the first to be of major practical importance. PL/I has capabilities derived from FORTRAN, COBOL, and ALGOL but has not (yet?) succeeded in one of its implicit objectives which was to replace these languages; it was preceded by JOVIAL in the attempt to combine capabilities for several application areas. SIMSCRIPT built on all the previous discrete simulation languages. SNOBOL was a good but fairly obvious improvement to the concepts introduced in COMIT.

3. Chronological Development

In the first part of this section the major earlier periods of time are delineated and their language highlights indicated. This spotlights and supplements information appearing in the chart. The second part of this section recognizes that in the field of programming languages there are key *concepts* which are different from the mere development of languages. The third part of this section discusses the languages and topics which are of current interest and activity.

3.1 Major Periods of Earlier Time

3.1.1 Earliest Years: 1952–1956. This period was a time of preliminary groping and of attempting to understand the concepts and limitations of programming languages. The terms pseudo-code, automatic coding, automatic programming, compiler, and interpreter were common. In today's terminology, pseudo-code merely means a language other than the normal machine code, automatic coding was the general process of writing in a higher level language, and automatic programming was intuitively felt to be something higher than automatic coding. It is interesting to note that the term automatic programming is coming back into vogue after many years of dormancy, but with a somewhat wider meaning.

Of all the languages developed in this time period, only FORTRAN (intended for numerical computation) and APT (for machine tool control) have lasted till today, and both have undergone numerous revisions.

3.1.2 Milestone Meeting: 1956. The first major meeting held solely to discuss higher level languages—or automatic coding as the subject was referred to then—was at the Franklin Institute in 1956. At the time of this meeting, the battle for the use of higher level languages was by no means won. No systems were in wide use; FORTRAN had not been released (in fact it was not discussed at the meeting, although it was presented at the 1957 Western Joint Computer Conference). FLOWMATIC was in use in essentially its prototype form by (only) one company. The other systems were essentially being used only by their developers.

At this meeting at the Franklin Institute, the following languages were described and presented:

B-0 (FLOWMATIC). The first English-like language for business data processing problems. Planned and implemented (only) on the UNIVAC I.

PRINT I. Really a powerful three address pseudocode, but of significance because it provided capabilities for mathematical problems on a machine which was basically designed for business data processing, namely the IBM 705.

OMNICODE. Really a type of assembly language in spirit and format but with powerful operations for scientific and commercial computation. Designed for the IBM 650 and 702 with a strong concern for conversion potential.

IT. A language for mathematical problems which was awkward in notation because of the limited character set of its intended machine, the IBM 650. Significant because it was intended for use on a small machine.

Matrix Compiler. A higher level language containing operations for doing matrix computations. Designed for the UNIVAC I. Significant because it is one of the first languages for specialized application areas (considering matrix manipulation as a specialized area).

NCR 304. This is obviously a machine and not a language. It is significant because it was apparently the first attempt to develop a computer which would make "automatic coding" unnecessary because the order code was at high enough level. (Although the machine was apparently a success, it did not eliminate higher level languages.)

3.1.3 Most Prolific Years: 1958–1959. It seems clear that the two most significant years in the history of programming languages are 1958 and 1959. The following events all occurred during that period:

1. The development and publishing of the IAL (International Algebraic Language) report, which became known as ALGOL 58.
2. The development of three languages based on the IAL specifications, namely NELIAC, MAD, and CLIP (which eventually was the foundation for JOVIAL). NELIAC, MAD, and JOVIAL were in use at least as late as 1971, with the latter primarily used in military applications.
3. The presentation to the 1959 UNESCO meeting of J. Backus' formalism for describing ALGOL [5]. This was the foundation for much of the theoretical work done in programming languages since then.
4. The formation in May 1959 of the CODASYL Short

Languages from 1960 Tower of Babel
Cover of *Communications of the ACM,* Vol. 4, No. 1, January 1961.†

ABC
ACT
ADES II
AIMACO
ALGO
*ALGOL
ALTAC
ALTRAN
API
APS
APS III
*APT
APX III
ARGUS
BACAIC
BALGOL
BIOR
CAGE
CL-I
CLIP
*COBOL
COLASL
*COMIT
COMMERCIAL TRANSLATOR
FACT
FLEXIMATIC
FLIP
FLOWMATIC
FORAST
*FORTRAN
FORTRAN II
FORTRAN III
FORTRANSIT
GAT
GOOFUS
GP
IPL
*IPL V
IT
IVY
*LISP
*MAD
*MADCAP
MAGIC
MATHMATIC
MCP
MISHAP
MYSTIC
*NELIAC
NUIT
9 PAC
PACT I
PRINT
SALE
SAP
SHADOW III
SLANG
SMAC
SOAP
SOS
STRAP I
STRAP II
SURGE
TAC
THREADED LISTS
TRIE
UNCOL
UNICODE
USE
VIPP
X1
X2
XTRAN

Languages from 1967 Tower of Babel
End paper in *Programming Languages: History and Fundamentals,* J.E. Sammet, Prentice-Hall, Inc., 1969.††

A-2 & A-3
ADAM
AED
AESOP
AIMACO
*ALGOL
ALGY
ALTRAN
AMBIT
AMTRAN
Animated Movie
APL
APL\360
*APT
BACAIC
BASEBALL
BASIC
BUGSYS
C-10
CLIP
CLP
*COBOL
COGENT
COGO
COLASL
COLINGO
*COMIT
Commercial Translator
Computer Compiler
Computer Design
CORAL
CORC
CPS
Culler-Fried
DAS
DATA-TEXT
DEACON
DIALOG
DIAMAG
DIMATE
DOCUS
DSL/90
DYANA
DYNAMO
DYSAC
English
Extended ALGOL
FACT
FLAP
FLOW-MATIC
FORMAC
Formula ALGOL
*FORTRAN
FORTRANSIT
FSL
GAT
GECOM
GPL
GPSS
GRAF
Graphic
ICES
IDS
Information Algebra
*IPL-V
IT
JOSS
JOVIAL
Klerer-May
L^6
Laning and Zierler
LDT
Lincoln Reckoner
*LISP 1.5
LISP 2
LOLITA
LOTIS
*MAD
*MADCAP
Magic Paper
MAP
MATHLAB
MATH-MATIC
Matrix Compiler
META 5
MILITRAN
MIRFAC
*NELIAC
OCAL
OMNITAB
OPS
PAT
PENCIL
PL/I
PRINT
Proposal Writing
Protosynthex
473L Query
QUIKTRAN
SFD-ALGOL
Short Code
SIMSCRIPT
SIMULA
Simul. Dig. Syst.
SNOBOL
SOL
Speedcoding
SPRINT
STRESS
STROBES
Symbolic Math. Lab.
TMG
TRAC
TRANDIR
TREET
UNCOL
UNICODE

*These 10 languages are the only ones appearing in all three lists. In the case of ALGOL, COMIT, FORTRAN, and LISP, the same version number does not show and/or apply on all three lists, but there is enough similarity to justify considering them as common to all three lists.

†Some items in this tower are not really languages but have been included to provide a complete list of the tower contents.

††Some of these languages were only considered of historical interest in 1967 and were not in use at that time.

LANGUAGE HISTORY CHART

First letter of each name has been aligned with the approximate date on which work began.

THIS TYPE STYLE indicates languages of major importance, because of their wide usage or technical significance.

THIS TYPE STYLE indicates languages of moderate importance.

THIS TYPE STYLE is used for all other languages.

Parentheses were used to indicate alternate names, or the later addition of the sequence number "1."

(double line) indicates that the second language is a direct extension of the first

(solid line) indicates that the second language is an approximate extension of the first, i.e., very similar to the first, but not completely upward compatible

(long dashes) indicates strong influence; sometimes the second language is "like, or in the style of" the first

(short dashes) indicates an approximate subset

Each of the following marks is associated with the language above or to its left:

- ● indicates preliminary or informal specifications or manual
- ■ indicates a public manual, or formal publication via technical paper, or public presentation
- ▲ release for usage outside development group

General Comments

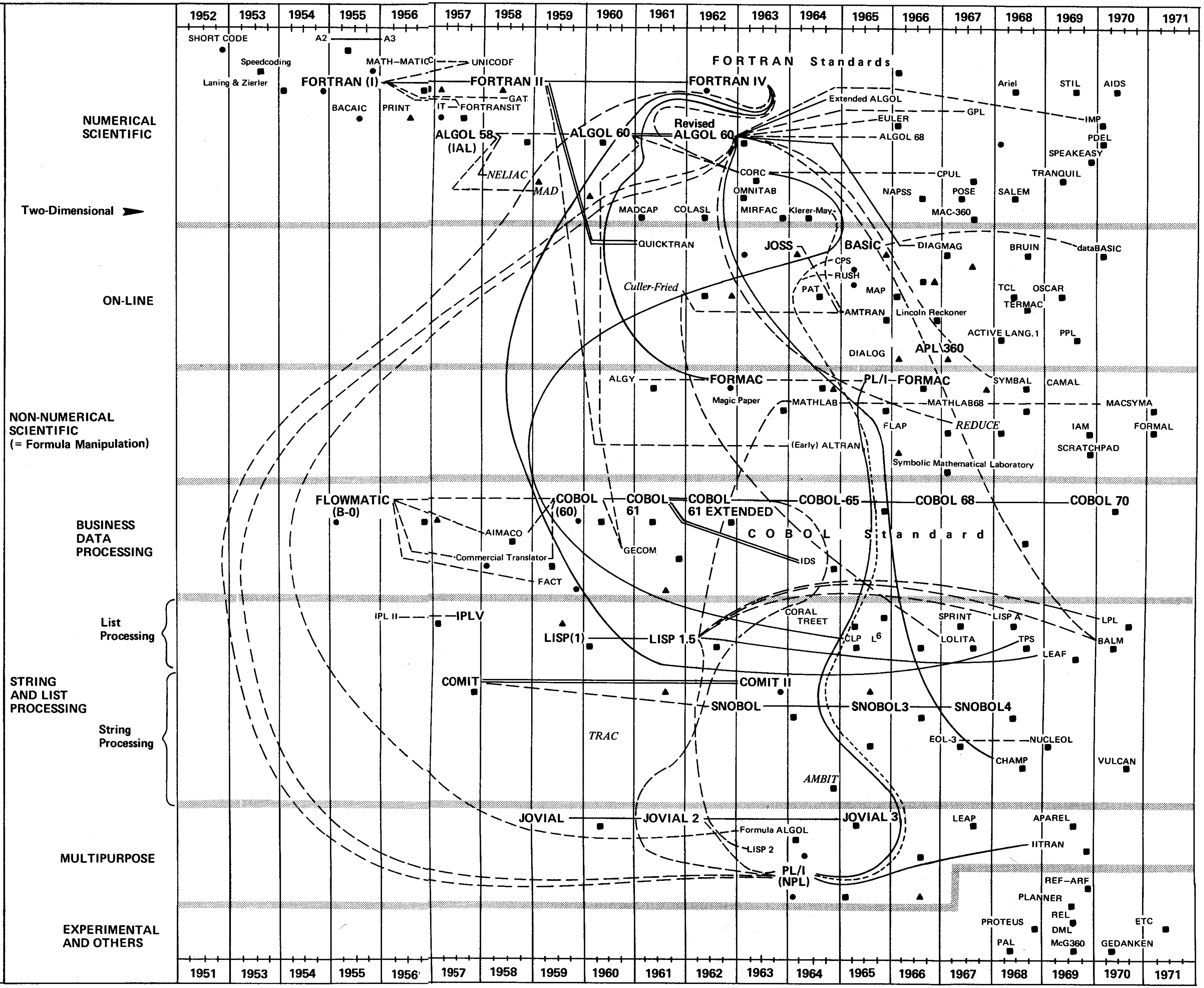
This chart represents only the personal opinions of the author as far as value judgments are involved, and the author's best estimate in many cases as far as dates are involved. The indications of the start of the work are the most questionable.

The information for languages in 1971 is based solely on those listed in "Roster of Programming Languages–1971," Computers and Automation, Vol. 20, No. 6B (June 1971), pp. 6-13.

In most cases, dialects with differing names have been omitted. This has the unfortunate effect of appearing to minimize the importance of some languages which spawned numerous versions under differing names (e.g., JOSS).

Languages for specialized application areas (e.g., simulation, machine tool control, civil engineering, systems programming) have not been included because of space considerations. This explains the absence of such obvious languages as APT, GPSS, SIMSCRIPT, COGO, BLISS.

Acknowledgment: The idea for such a chart in such a format came from the one by Christopher J. Shaw entitled "Milestones in Computer Programming" and included with the [ACM Los Angeles Chapter] SIGPLAN notices, February 1965.



"Programming Languages: History and Future"
by Jean E. Sammet
Communications of the ACM, Vol. 15, July 1972

Languages in 1971 Roster

From "Roster of Programming Languages," J.E. Sammet, *Computers and Automation,* Vol. 20, No. 6B, June 1971.

ACTIVE LANGUAGE I
AED
AESOP
AIDS
ALADIN
*ALGOL 60
ALGOL 68
ALTRAN
AMBIT
AMTRAN
Animator
APAREL
APDL
APL\360
*APT
Ariel
ATLAS
ATOLL
B-LINE
BALM
BASIC
BCPL
BLISS
BRUIN
BUGSYS
CAMAL
CATO
CCL
CESSL
CHAMP
*COBOL
COGENT
COGO
COIF
*COMIT II
Computer Animation Language
Computer Design Language
COMSL
COMTRAN
CORAL
CSS/II
Coursewriter
Coursewriter III
CPS
CSMP
CSSL
CTL
CUPL
Cypher Text
DARE
dataBASIC
Data Structures Language
DATA-TEXT
DCDL
DGL
DIALOG
DIMATE
DML
DSL
DYNAMO II
ECAP II
ELP
EOL-3
ETC
EULER
Extended ALGOL
FLAP
FOIL
FORMAC
FORMAL
*FORTRAN
FSL
GAN
GASP
GEA
GEDANKEN
General Purpose Graphic Language
GPSS
GRAF
GRAIL
Graphic Language
GRIND
HINT
IAM
ICES
IDS
IITRAN
IMP
*IPL-V
JOSS
JOVIAL
Klerer-May
L^6
LEAF
LEAP
Lincoln Reckoner
*LISP 1.5
LISP A
Logic Design Language
Logo
LPL
LRLTRAN
LSYD
MAC-360
MACSYMA
*MAD
*MADCAP
MARSYAS
MATHLAB 68
McG360
MENTOR
META 5
MOBSSL-UAF
NAPSS
*NELIAC
NPPL
NUCLEOL
OLDAS
OMNITAB II
OSCAR
PAL
PDEL
PIRL
PL/I
PL/I-FORMAC
PLACE
PLANIT
PLANNER
PPL
PREP
PROTEUS
REDUCE
REF-ARF
REL English
RTL
RUSH
SALEM
SCRATCHPAD/1
SCROLL
SIMSCRIPT 1.5
SIMSCRIPT II
SIMULA
SLANG
SNAP
SNOBOL4
SPEAKEASY
SPRINT
STIL
STRESS
STROBES
STRUDL
SYMBAL
TCL
TERMAC
TMG
TPS
TRAC Language
TRANQUIL
TRANS
TREET
TROLL
VULCAN
WRITEACOURSE
XPL

Range Committee (later renamed the COBOL committee) and the completion of the specifications by December 1959 (although they were not published until 1960).
5. The development and availability of language specifications for AIMACO, Commercial Translator, and FACT.
6. The start of work on the development of LISP in 1959.
7. The first implementation of COMIT; (a brief description of the language appeared as early as December 1957).
8. The start of work on JOVIAL in 1959.
9. The availability of a running version of IPL-V in early 1958 on the IBM 650; a new version was operational on the IBM 704 at the end of the summer in 1959.
10. The development of a *second* version of APT, namely APT II for the IBM 704. (See also item 11.)
11. The development of several (other) languages for specialized areas, e.g. DYANA (1958); DYNAMO (1959); work started on AED (1959).

3.1.4 1960-1970. This decade saw a maturation of the programming language field. During this time the battle over the use of higher level languages was clearly won in the sense that machine coding had become the exception rather than the rule. Even the development of systems programs using higher level languages is fairly well accepted [3]. The use of powerful macro systems, e.g. ETC [7] and "half way" languages such as PL/360 [22] provided some of the advantages of higher level languages but without any attempt to be machine-independent.

The major new languages were ALGOL, COBOL, and PL/I, of which only the last two were significantly used in the United States. While ALGOL 68 was defined, its implementation was just starting around 1970.

The advent of time-sharing brought a host of online languages, starting with JOSS and later followed by BASIC, which became very widely used. Each had many imitators and extenders. APL\360, made available late in the 1960s, became popular among certain specific groups.

The development of higher level languages for use in formula manipulation was triggered by FORMAC and Formula ALGOL, although only the first has been widely used. String processing and pattern matching became popular with the advent of SNOBOL.

The simulation languages GPSS and SIMSCRIPT made this tool available to most users and also encouraged the development of other simulation languages.

A number of languages for specialized application areas continued to be developed. See [14, 18] for more information on this.

Perhaps one of the most important practical developments, although scorned by many theoreticians, was the development of official standards for FORTRAN and COBOL, and the start of standardization for PL/I.

3.2 Key Concepts Related to Programming Languages

To review the work of 20 years in programming languages requires the separation of the historical development of *specific languages* from the development of certain *concepts* which can affect all programming languages. In this context, these concepts are entirely different from concepts *in* the language. In approximately chronological order, I consider the major conceptual developments to be: (1) formal syntactic notation, (2) formal semantic definitional techniques, (3) attempts at designing machines whose instruction code is a higher level language, and (4) user defined languages. The reasons for their importance are indicated below. I have not listed either the concept of a higher level language or the concept of a compiler because they are so basic they can be taken for granted.[2]

The idea of a *formal syntactic notation* for a programming language was introduced in 1959 by Backus [5]. It is important to note that BNF is merely one maniestation of a formal syntactic notation and not the only one; e.g. the definition of COBOL uses a different but equally valid metalanguage, although this fact is often not realized or it is ignored. The significance of a formal syntactic notation is threefold:
1. It provides a rigorous way of defining the syntax of a language and eliminating the annoying ambiguities of sentences such as "a name contains six characters and both the first and last character cannot be a hyphen."
2. It provides the link between the practical concerns in the programming language field and the theoretical work done by linguists, most notably N. Chomsky. This permits application of linguistic concepts and techniques to programming languages. Although BNF was developed independently by Backus, it is really one of Chomsky's grammars with a different notation.
3. It led to the development of syntax directed compilers, which in turn gave rise to a more theoretical study of compilation techniques and the possibility of a "production line approach" to developing new compilers.

Formal semantic definitional techniques represent considerably more than just the next step beyond the formal syntactic definition. While much of the original concept was based on work of McCarthy [10], the first actual development of the formal semantic definition of a major language was done by the IBM Vienna Laboratory [9]. The monumental size of the formal definition of PL/I precludes it from being of current practical use. However, it is a major conceptual part of a currently unsolved problem in *all* of programming which is how to determine whether a program does what we want it to do. Enormous progress is being

[2] The concept of list processing does not appear here because it is fundamentally a programming technique which has been represented or included in languages. Also there is no discussion of the major concepts in implementation.

made in this area (see for example [1]), and there are two facets of concern to programming languages: to make sure the compilers translate correctly and to see what must be done to guarantee the correctness of a source program. Formal semantic techniques give a handle on solutions to both these problems.

The attempt at designing computers whose instruction code is a higher level language is a significant concept—although not yet achieved—because the whole field of programming languages is a clear manifestation that all the computers designed to date are unsatisfactory verbal communication devices. In other words, we have not solved the problem of how to bridge the gap between what the person wants to say about solving his problem and the physical circuits in the machine.

User defined languages have been an area of interest for many years. Only a few primitive attempts have been made; they are described briefly in [18]. Allowing people to define their own languages is a significant concept because it takes the control of "what is good for the user" out of the hands of language developers and puts it in the hands of the users.

3.3 Current Status and Topics

To sensibly look forward after looking backward we must first see where we are today. This section attempts to pinpoint what seems to be the current status and major topics in programming languages in 1972 (exclusive of implementation). The potential effect of these on the future will be discussed in Section 4.

Debates on specific *current languages* continue, with the most frequent arguments occurring about PL/I, ALGOL 68, and APL\360. The fact that the bulk of programming is done in languages *other* than these does not inhibit the debate about them. The arguments—at least in public—tend to come more from language developers and theoreticians than from the users. The latter continue to be (rightly) concerned about issues of cost effectiveness and compatibility and hence tend to stay with FORTRAN and COBOL.

The most astounding fact is that the proliferation of languages continues. There is no evidence that fewer languages are being created. The new languages that have been developed and/or used in the United States during the last few years are shown in Sammet [15, 16, 19].

Extensible languages—at least among researchers—continue to be of interest, but without anything of major significance to show for the years of work which have been involved. Contrast the contents of [4] with those of [2].

Proving correctness of programs is a topic receiving increased attention as evidenced by [1]. This concept is directly related to programming languages because most of the techniques being suggested involve changes or additions to programming languages, or the insertion of completely different kinds of statements in source programs to enable correctness proofs to operate.

The use of higher level languages for systems programming has finally been recognized as being both legitimate and practical. After almost 15 years of debate and negative views, an increasing number of systems programs are actually being written using higher level languages; they are described in [3]. The arguments now tend to be centered around *which* language should be used rather than *whether* one should be used.

4. Future Developments

Programming languages—using any definition—are the primary means by which a person communicates with a computer. Viewed from that perspective, all future developments should be aimed at making this communication easier and more cost effective. Each person doing research and development feels he has *the* answer. The real truth seems to be that there is no single best way for people to communicate with a computer, and hence no single solution. *Thus there will be no single language useful to everyone.*

This section is divided into three areas—broad concepts, specific techniques, and a brief discussion about the effect of current education about computers. Comments on the future of the current topics discussed in Section 3.3 are included in this section.

4.1 Broad Concepts

The major broad concepts that we should expect to see in the future are: (1) use of natural language (e.g. English), (2) user defined languages, (3) nonprocedural and problem defining languages, (4) an improvement in the user's computing environment, and (5) new theoretical developments.

The ultimate ease of communication with the computer allows the user to specify his instructions—or wishes—in a *natural language* such as *English.* That is a shorthand way of saying that the user could use his native language, including notation (e.g. algebra, molecular diagrams) appropriate to his particular field. This concept *does not* envision one single computer system understanding all of English. It *does* envision many systems, each of which is capable of dealing with a particular field, specifically including its specialized jargon. This is identical in concept to the communication between people which works fairly well providing both are in the same field and understand the same terms. The recognition of the importance of jargon serves as the main counterargument to the ambiguity problem which exists primarily *across* specialized vocabularies. By envisioning a series of semispecialized systems, we can eliminate the need for a single giant computer system which understands all of Webster's dictionary *plus* a large encyclopedia.

The arguments for and against this concept have been presented many times in print. References can be found in [14, Ch. XI].

Until we reach the ultimate situation described just above, the next best thing will be *user defined languages.* By this we mean (software) systems which permit users, first, to define languages that fit their own needs with respect to functional capability, jargon, and personal tastes in style, and then to easily implement them. The key part of the problem is to provide a system which permits easy implementation with an acceptable level of efficiency. In spite of numerous attempts going back many years, this development is still in a primitive stage. References and a further discussion of this can be found in Sammet [18] and in Thompson and Dostert [21].

This author has often stated that the term *non-procedural language* is a relative term which changes as the state of the art changes. Thus, as we develop languages (and compilers for them) which specify less detailed information, we increase the level of non-proceduralness. Such a major potential development can be accomplished either separately or combined with the user defined languages and/or *problem defining languages.* Ideally, the user would state only the definition of his problem and the computer system would develop the solution. While the day of asking the computer to "COMPUTE THE PAYROLL FOR MY COMPANY" is at least one or two decades in the future, I believe we will see a large decrease in the amount of detail a user must provide. More specifically, I expect more statements about *what* is to be done and fewer details on *how* to do it. There will be compilers which can effectively determine which of many alternative algorithms should be used in a given situation. Examples of this kind of research include ISDOS (Teichroew and Sayani [20]), NAPSS (Rice [11]), and the goal oriented PLANNER (Hewitt [8]).

With today's powerful computers, the *user's computing environment* is large, complex, and needs improvement. When the user writes his program he has just started on the communication problem. He must now interface with command languages to the operating system, debug by pouring over cryptic diagnostic messages received about his errors, and try to determine his costs in an installation when he doesn't know who else is using the computer at the same time he is. In the future when we have practical networks of computers, he may not even know which computer he will be using! All of these difficulties tend to negate the advances made in easing the direct statement of the user's problem, which is represented by the program in some higher level language. In the present environment the latter represents only the tip of the iceberg, and we must certainly make the rest of the iceberg easier to use.

Virtually everyone agrees that today programming is an art, not a science. Many—although somewhat fewer—people would contend that programming can (and should) be made into a science. I think that this is feasible, but it requires many *theoretical developments.* These range from better methods of defining programming languages, to techniques enabling us to know whether any program does what we really want, to the advances in understanding English, which are required to reach the first stated goal.

4.2 Specific Future Developments

There are a number of specific items on which comments should be made pertaining to their role in the future. To a large extent, these parallel the major topics of interest today.

The *use of current languages* is not likely to change drastically for many years. It seems likely that FORTRAN and COBOL will be with us for at least five and probably ten more years. The potential standardization of PL/I will probably wean away some FORTRAN and COBOL adherents, but by now the investment in the earlier languages is so large that the advantages of using only one language for most practical problems cannot counteract the past history. ALGOL 68 will probably play the same role that ALGOL 60 did—namely inspire another round of development of implementation techniques and devotion to elegance in language with less concern for the practical world. It seems virtually certain that ALGOL 68 will *not* become widely used in the United States, and probably not elsewhere. APL\360 has fanatic adherents, but in spite of their hopes and claims it does not seem likely that this language (or system) will replace all other languages.

For the foreseeable future *languages for specialized application areas* will continue to proliferate for reasons well described in [21]. In essence they indicate that there will be large economic advantages accruing to a user from (1) a language which is efficient because it deals only with matters of direct concern in a particular application and (2) algorithms which are inherently known and available within a specific problem domain; these specialized languages will also continue to increase in number and usefulness as the technology improves in the broad concept of providing techniques which allow the user to easily define and implement his own language. As one subclass, the application area of systems programming will see the continued development of improved languages which simultaneously provide most advantages of a higher level language with minimum machine dependence.

The development of *extensible languages* has retrogressed, or at best stood still, in the past few years. For the future I hope—but think it unlikely—that this area will do another turnaround and become a useful tool. It should be recognized however that extensible languages are primarily a means to an end rather than a goal in themselves. The end which they help achieve is the ability for people to easily define their own languages.

4.3 Effect of Increased Education

Computers are now becoming common—if not yet commonplace—in high schools, and have become frequent even in elementary schools. To determine the real effect—both direct and subliminal—of this education on the next generation requires a better crystal ball than this author possesses. However, it would seem that at the very minimum the following things will happen: (1) people will grow up recognizing the computer as a tool for everyone, just as the automobile, the telephone, and the stove are; (2) more people will learn to use the computer—they probably will be taught languages like BASIC, FORTRAN, and COBOL, even though these do not represent the best technology we have for programming; and (3) the use of computers will increase as the result of more understanding of what they can be used for.

5. Summary

There have been over 200 higher level languages developed in the 20 years between 1952 and 1972. Of these languages, only 13 are deemed of major significance from either a conceptual and/or a usage viewpoint. The two years 1958 and 1959 were by far the most significant in the past 20 years, and most of the programming language activities of 1972 result directly or indirectly from work done in those two years. Current *discussions* tend to involve PL/I, ALGOL 68, and APL\360 while most of the *productive work* is done in FORTRAN and COBOL. The major topics of 1972 are extensible languages, proving correctness of programs, the use of higher level languages for systems programming and the continued proliferation of languages for specialized application areas. For the future, broad conceptual developments to be expected include the use of English for programming, user defined languages, more nonprocedural and problem defining languages, improvement of the user's computing environment, and new theoretical developments. Specific future developments will most likely include major continued use of FORTRAN and COBOL, and continued proliferation of languages for specialized application areas. The effect of the education about computers being supplied to high school and elementary school students will be so profound that a reasonable prediction seems impossible.

References

1.
ACM. *Proceedings of an ACM Conference on Proving Assertions About Programs. SIGPLAN Notices 7*, 1 and *SIGACT News* 14 (Jan 1972).

2.
ACM SIGPLAN. *Proceedings of the Extensible Languages Symposium. SIGPLAN Notices 4*, 8 (Aug. 1969).

3.
ACM SIGPLAN. *Proceedings of a SIGPLAN Symposium on Languages for Systems Implementation. SIGPLAN Notices 6*, 9 (Oct. 1971).

4.
ACM SIGPLAN. *Proc. ACM SIGPLAN Conf. on Extensible Languages. SIGPLAN Notices 6*, 12 (Dec. 1971).

5.
Backus, J.W. The syntax and semantics of the proposed international algebraic language of the Zurich-ACM-GAMM Conference. Proc. International Conf. Information Processing, UNESCO, Paris, 1959, R. Oldenbourg, Munich; Butterworth, London, 1960, pp. 125–32.

6.
Bemer, R.W. A politico-social history of ALGOL. In *Annual Review in Automatic Programming*, Vol. 5. M. Halpern and C. Shaw (Eds.), Pergamon Press, New York, 1969, pp. 151–237.

7.
Dickman, B.N. ETC –An extendible macro-based compiler. Proc. AFIPS 1971 SJCC, Vol. 38, AFIPS Press, Montvale, N.J. pp. 529–538.

8.
Hewitt, C. Procedural embedding of knowledge in Planner. Proceedings of Second International Joint Conference on Artificial Intelligence, British Computer Society, London, 1971, pp. 167–184.

9.
Lucas, P., and Walk, K. On the formal description of PL/I. In *Annual Review in Automatic Programming*, Vol. 6, 3. Pergamon Press, New York, 1969, pp. 105–182.

10.
McCarthy, J. A formal description of a subset of ALGOL. In *Formal Language Description Languages*, T.B. Steel Jr. (Ed.), North-Holland Pub. Co., Amsterdam, 1965, pp. 1–7.

11.
Rice, J.R. On the construction of polyalgorithms for automatic numerical analysis. In *Interactive Systems for Experimental Applied Mathematics*. M. Klerer and J. Reinfelds (Eds.). Academic Press, New York, 1968, pp. 301–313.

12.
Rosen, S. Programming systems and languages—a historical survey. In *Programming Systems and Languages*, S. Rosen (Ed.), McGraw-Hill, New York, 1967, pp. 3–22.

13.
Rosen, S. Programming systems and languages—some recent developmens. In *Programming Systems and Languages*, S. Rosen (Ed.), McGraw-Hill, New York, 1967, pp. 23–26.

14.
Sammet, J.E. *Programming Languages: History and Fundamentals*. Prentice-Hall, Englewood Cliffs, N.J. 1969.

15.
Sammet, J.E. Roster of programming languages, 1970. *Computers and Automation 19*, 6B (Nov. 1970), 6–11, 21.

16.
Sammet, J.E. Roster of programming languages, 1971. *Computers and Automation 20*, 6B (June 1971), 6–13.

17.
Sammet, J.E. Problems in, and a pragmatic approach to, programming language measurement. Proc. AFIPS 1971 FJCC, Vol. 39, AFIPS Press, Montvale, N.J., pp. 243–251.

18.
Sammet, J.E. An overview of programming languages for special application areas. Proc. AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., 299–311.

19.
Sammet, J.E. Roster of programming languages—1972. (In preparation.)

20.
Teichroew, D., and Sayani, H. Automation of system building. *Datamation 17*, 16 (Aug. 1971), 25–30.

21.
Thompson, F.B., and Dostert, B.H. The future of specialized languages. Proc. AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., pp. 313–319.

22.
Wirth, N. PL360, A programming language for the 360 computers. *J. ACM 15*, 1 (Jan. 1968), 37–74.