When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-1960

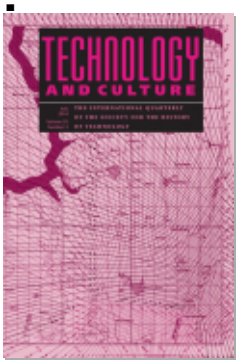
Nofre, D.; Priestley, M.; Alberts, G.

Published in:
Technology and Culture

DOI:
10.1353/tech.2014.0031

When Technology Became Language: The Origins of the Linguistic Conception of Computer Programmi 1950–1960

David Nofre, Mark Priestley, Gerard Alberts



Technology and Culture, Volume 55, Number 1, January 2014, pp. 40-75 (Article)

Published by The Johns Hopkins University Press
DOI: 10.1353/tech.2014.0031

When Technology Became Language

The Origins of the Linguistic Conception of Computer Programming, 1950–1960

DAVID NOFRE, MARK PRIESTLEY, and GERARD ALBERTS

Introduction

The second half of the 1950s saw the emergence of a new vision of how computers were to be programmed. At the beginning of the decade, programmers had to express the instructions for solving a problem in obscure numerical codes that were different for each machine. By the decade's end, however, they could write programs that included familiar mathematical formulas, and, in some cases, even expect the same program to run on different machines, thanks to the development of systems like FORTRAN and IT. Furthermore, professional and industrial bodies were putting forward ambitious proposals for very powerful "programming languages," as the codes were now widely called, and some of these, notably ALGOL and COBOL, were explicitly defined to be machine-independent notations. In

David Nofre is associated with the Centre d'Estudis d'Història de la Ciència, Universitat Autònoma de Barcelona; Mark Priestley is an independent researcher based in London; and Gerard Alberts is an associate professor of the history of mathematics and computing at the Korteweg-de Vries Institute for Mathematics, University of Amsterdam. Nofre and Alberts's contribution was developed as an element of the Software for Europe project, as part of the European Science Foundation Eurocores Program "Inventing Europe," and co-funded by the Netherlands Organization for Scientific Research (NWO 231-53-004). Research for this article was assisted by the award to Nofre of a 2010 Lemelson Center Travel to Collections Award from the Smithsonian Institution and a 2009 Arthur L. Norberg Travel Award from the Charles Babbage Institute. The authors thank Eden Medina for helpful comments on an early draft of this article; Matthias Dörries, Helena Durnová, Hans Dieter Hellige, Janet Martin-Nielsen, and Edgar Daylight for insightful comments on its early ideas; and the three anonymous referees and Suzanne Moon for providing constructive comments and suggestions. They also thank Peggy Aldrich Kidwell for access to materials in the Computer Documentation Collection at the National Museum of American History, Smithsonian Institution, Washington, D.C.

0040-165X/14/5501-0002/40–75

parallel with this transformation, researchers began to conceive of programs and programming languages as being largely autonomous from the machines; they were no longer seen as mere tools, but became available objects of knowledge in their own right, a fact of great significance for the emergence in the 1960s of the software industry and the discipline of computer science.

In this article, we ask how it came to seem natural to think of programming highly complex electronic devices as a linguistic activity, and we give an account of the processes that led up to this conceptual transformation.[1] At the core of the argument lies a shift in the metaphorical use of the term *language* in connection with computers that took place in the mid-1950s in the United States. With roots in the fascination with robots that pervaded American culture before and during World War II, the language metaphor entered modern computing vocabulary at the end of the war as part of a cybernetic discourse that described modern computers as if they were semi-autonomous, almost human-like agents.[2] Early volumes on computing machinery, such as E. C. Berkeley's *Giant Brains, or Machines That Think* (1949) and B. V. Bowden's *Faster Than Thought* (1953), expressed the same fascination.[3] The new automatic calculators seemed to embody a form of intelligence and language was considered to be an intrinsic attribute of these agents, on a par with their material components. Specialists like Arthur Burks and George Stibitz would describe programming as a form of communication, involving translation from human language into "the language the machine can understand."[4] During the second half of the 1950s, however, the language metaphor lost its anthropomorphic connotation and acquired a more abstract meaning, closely related to the formal languages of logic and linguistics, in addition to greater epistemic impact.[5] We argue that

1. It is beyond the scope of this article to discuss the suitability of the use of the language metaphor in computer programming, and the relation among programming languages, formal languages, and natural languages.

2. Chihyung Jeon, "Flying Weather Men and Robot Observers"; Bernadette Longo, "Metaphors, Robots, and the Transfer of Computers to Civilian Life." For a contrast with the more stoical, even fearful, British accounts in the press, see Angus McLaren, *Reproduction by Design*; and Mark D. Bowles, "US Technological Enthusiasm and British Technological Skepticism in the Age of the Analogue Brain."

3. Edmund C. Berkeley, *Giant Brains, or Machines That Think*; B. V. Bowden, ed., *Faster Than Thought*.

4. The expression is quoted from the preface of Arthur W. Burks, Herman H. Goldstine, and John von Neumann's *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. A few months earlier, the same expression was used in an anonymous article featured in *Science News-Letter* describing Vannevar Bush's wartime work at MIT on differential analyzers; see "Mathematical Machine." This understanding of the language metaphor came to be embedded in the expression *machine language*, which was in use as early as January 1947; see George R. Stibitz, "A Manual of Operation for the Automatic Sequence Controlled Calculator," 59.

5. In this article, we focus on the process of the acquisition of explanatory or epistemic power by the language metaphor, rather than on its capacity to shape thought or

JANUARY
2014
VOL. 55

this transformation was related to the appearance of the commercial computer in the mid-1950s, and in particular, to the increasing heterogeneity of computer installations at the time. The early users of the term *programming language* were principally computer-user groups and computer-installation managers attempting to bring about the cross-machine compatibility of programs; common or universal notations would facilitate the exchange of programs among and within organizations, and would also provide a suitable vehicle for teaching programming in universities. Managers and educators thus no longer found it helpful to think of programming notations as attributes of individual machines, and began to draw on the disciplines of symbolic logic and linguistics to develop models of intelligibility that would enable abstraction away from the machine and toward the development of free-standing notations. This process reached its climax with the development, between 1958–60, of ALGOL for scientific computation and COBOL for data-processing. During the next decade, linguists appropriated concepts developed to study programming languages and applied them to the analysis of natural languages, at the same time as molecular biologists found that programming metaphors opened up new avenues of research, such as the interpretation of DNA as "code" or "language."[6]

Early attempts to write the history of programming languages centered around accounts of the development of individual languages, with a strong focus on technical details.[7] Later historiography has taken a wider perspective, situating the emergence of programming languages in the 1950s within the general context of the evolution of computing, and viewing it as a response to the mounting complexity and rising costs of the programming process.[8] Meanwhile, historical studies on the origins of artificial

---

discourse. For a discussion of this, see Matthias Dörries, ed., "Language as a Tool in the Sciences." With regard to formalization, a similar process took place in other disciplines, such as economics, philosophy of science, linguistics, and psychology, as part of the rise of formal logic as one of the main pillars of the U.S. cold war regime of knowledge. See Philip Mirowski and Esther-Mirjam Sent, "The Commercialization of Science and the Response of STS," table 1; and Dominique Pestre, *Science, Argent et Politique*, chap. 1.

6. William John Hutchins, *Machine Translation*; Lily Kay, *Who Wrote the Book of Life?*

7. ACM, *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*; Thomas J. Bergin and Richard G. Gibson Jr., *History of Programming Languages II*; Richard L. Wexelblat, *History of Programming Languages I*; Donald Knuth and Luis Trabb Pardo, "The Early Development of Programming Languages"; Jean E. Sammet, *Programming Languages*; Saul Rosen, "Programming Systems and Languages."

8. Nathan Ensmenger, *The Computer Boys Take Over*; Paul E. Ceruzzi, *A History of Modern Computing*; Martin Campbell-Kelly and William Aspray, *Computer*. Furthermore, recent trends include the approach to the history of programming languages from the user-centric perspective: Gard Paulsen, "Software in Telecommunications and the Programming Language Chili, 1974–1999"; or in the context of research funding policies: Jan Rune Holmevik, *Inside Innovation*.

intelligence, cognitive psychology, and formal structural linguistics—research areas strongly sponsored by the U.S. military—opened yet other perspectives.[9] These recent studies suggest a connection among these fields, the process of formalization of the language metaphor in programming, and the transformation of the computer into an information machine.

All of these approaches, however, take the *linguistic* nature of computer programming for granted. Indeed, it is surprising how rarely language appears in the list of relevant programming metaphors, despite periodic attempts to envisage program code as a form of literary expression.[10] It is as if we have become so accustomed to think of programming languages as *languages* that we forget that this analogy has its own history. In this article, we aim to open the black box of this metaphor to show how the very notion of a programming language emerged in connection with attempts to disembody programming knowledge as a response to the sudden increase in the variety of computers.

Many different interests came together in this complex transformation. Organizations began to look for ways of moving programs among machines of different types in order to protect and capitalize on their prior investment in programming. This process gained institutional support as representatives of computing facilities from the aircraft industry and the military began to form user groups to search for ways of coping with the costs of operating different machines and upgrading to new models. At the same time, staff members in academic computing centers worked to establish a space within which the free exchange of information and collaboration associated with military funding could be sustained as the development of computers became the preserve of large commercial companies.

All these trends led to an increased understanding of programming as a machine-independent activity, and, as a result, the language metaphor acquired an increasingly abstract nuance. This transformation cleared the way for the emergence, in the early 1960s, of computer science as a field and of software as a distinct entity. Programming languages became new epistemic objects that were no longer connected to a particular machine or

9. Ronald Kline, "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence"; Randy Allen Harris, "Chomsky's other Revolution"; Janet Martin-Nielsen, "Private Knowledge, Public Tensions"; Marcus Tomalin, *Linguistics and the Formal Sciences*; Hunter Crowther-Heyck, "George A. Miller, Language, and the Computer Metaphor of Mind"; Paul N. Edwards, *The Closed World*.

10. Note the absence of the language metaphor in Timothy R. Colburn and Gary M. Shute, "Metaphor in Computer Science"; Alan F. Blackwell, "The Reification of Metaphor as a Design Tool"; and Gerald J. Johnson, "Of Metaphor and the Difficulty of Computer Discourse." Even when language becomes the focus of the analysis, it is mostly in terms of human–machine communication; see Jörg Pflüger, "Language in Computing." On code as literary expression, see, for example, Donald E. Knuth, *Literate Programming*.

JANUARY
2014
VOL. 55

computer laboratory, thus enabling the nascent field of computer science to objectify its topics of study—namely, algorithms—and to develop a full disciplinary apparatus, including a conceptual toolkit, standard terminology, and an ever-increasing range of textbooks and other expository material.[11]

This article traces the genealogy of the language metaphor in programming, one of the most essential metaphors around which computer science has been built. In doing so, it reflects the growing awareness among historians of technology of the crucial role of language in shaping and mediating our understanding of technologies.[12] The emergence of programming languages entailed not only a transformation in the understanding of the activity of programming, but also in the way we think of the electronic digital computer. By abstracting away from the machine, programming languages and, more generally, software came to mediate our understanding of what a computer is, leading eventually to the conceptualization of the computer as an infinitely protean machine.[13]

The article also reflects the increasing interest in software history among scholars from a variety of disciplines, ranging from historians exploring the emergence of software as a distinct object and industry to media studies scholars trying to understand the meaning of the technology behind the visual culture of the new digital media. All are seeking to tease out the diversity of beliefs, artifacts, tools, and practices concealed behind the label *software*.[14] Similarly, the article shows how in the late 1950s, specialists in computer programming began to conceive of their activity as a proper and independent field of inquiry. In taking this step, they left behind the understanding of programming as a form of communication with the machine, involving the translation of a problem into a highly localized, machine-specific code.

11. Disembodiment is a crucial step in the constitution of a body of knowledge into proper science; see Steven Shapin, "The Mind Is Its Own Place." Note again the parallelism with the transformation of "chemistry" into a scientific discipline during the nineteenth century, and the important role played by language in such a transformation; see Bernadette Bensaude-Vincent, "Languages in Chemistry." Ensmenger has convincingly argued for the importance of the concept of an algorithm in the creation of the field of computer science, and its evolution toward a Kuhnian "normal science"; see Ensmenger, *The Computer Boys Take Over*.

12. Carroll W. Pursell, "Technologies as Cultural Practice and Production," 716–17. A similar point is made in Peter Galison, "The Ontology of the Enemy," 265.

13. We thank one of the anonymous reviewers for this observation.

14. Wendy Hui Kyong Chun, *Programmed Visions*; Ensmenger, *The Computer Boys Take Over*; Michael S. Mahoney, "What Makes the History of Software Hard"; Matthew Fuller, *Software Studies*; Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*; Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg, *History of Computing*; Thomas Haigh, "Software in the 1960s as Concept, Service, and Product."

"The Language the Machine Can Understand"

Automatic calculators have long been viewed in anthropomorphic terms by both their creators and the general public. In the nineteenth century, Charles Babbage apologized for using anthropomorphic language to describe his Analytical Engine, finding that its convenience and naturalness outweighed the possibly misleading connotations.[15] A century later, after the disclosure in the mid-1940s of wartime computing projects in the United States, the popular press, as well as the scientists and engineers involved in the building of computers, regularly described the new calculating devices as "robots" or "giant brains."[16]

The reemergence of this analogy has roots in the fascination with robots characteristic of U.S. society during the Great Depression. In this period, robots were a notable feature in the world's fairs of Chicago (1933–34) and New York (1939–40), being portrayed as the incarnation of the power of modern science and engineering.[17] Expressions like "mechanical brain," "mathematical brain," or even "robot" were already in use during the period to describe, for example, Vannevar Bush's differential analyzers at MIT and Hollerith punched-card equipment.[18] The development of cybernetics, which viewed natural and artificial control mechanisms as closely related species in the same genus, gave further strength to the analogy.[19] As early as the 1930s, Alan Turing was arguing that computing machines could be built to model features of the human brain, and in 1945, John von Neumann presented the design of an electronic computer using components that were explicitly modeled on biological neurons.[20]

Although contested, these anthropomorphic images were widespread and represented more than a convenient analogy;[21] they were based on a

15. Charles Babbage, "On the Mathematical Powers of the Calculating Engine," 31.

16. Longo, "Metaphors, Robots, and the Transfer of Computers to Civilian Life"; C. Dianne Martin, "The Myth of the Awesome Thinking Machine."

17. Aristotle Tympas, "From Digital to Analogue and Back"; David E. Nye, *Electrifying America*, 342, 390; Robert W. Rydell, "The Fan Dance of Science."

18. During the years 1930–45, one can find a significant number of articles in *Science News-Letter* wherein the term *robot* is used to describe all kind of automatic devices, including computing artifacts.

19. David A. Mindell, *Between Human and Machine*. Classic statements of the cybernetic point of view are Arturo Rosenblueth, Norbert Wiener, and Julian Bigelow, "Behavior, Purpose and Teleology"; and Norbert Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*.

20. Both Turing and von Neumann were closely involved in the early development of cybernetics; see William Aspray, "From Mathematical Constructivity to Computer Science," 147–48, 225.

21. Attempts were routinely made by computer-builders and applied mathematicians to downplay the significance of such analogies, and to reestablish the passive, tool-like nature of automatic computers. The tension between these two positions reverber-

JANUARY
2014
VOL. 55

detailed functional equivalence that showed how the new machines could replace the female human "computers" who had carried out the mechanical aspects of large-scale computational tasks up to the 1940s.[22] In 1952, Grace Hopper explicitly declared that the current aim was "to replace, as far as possible, the human brain by an electronic digital computer," and presented a general schema of the processes involved in carrying out a mathematical operation, showing how it could be instantiated equally well by a human or by a machine, such as Remington Rand's UNIVAC.[23]

As a result, early developers and users endowed the computer with forms of agency and autonomy, to a much greater extent than with other kinds of calculators. This had consequences for the ways that the uses of such machines, and in particular the tasks of coding and programming, were discussed. In 1947, for example, George Stibitz, a mathematician and developer of relay computers at Bell Labs, described computing machines as having the ability to "understand mathematical language" and to "interpret . . . symbolically represented operations in terms of the machine operations of which [they are] capable." He went on to describe the self-checking abilities of the computer as being like the kinesthetic sense possessed by living organisms. In particular, writers trying to convey a sense of the capabilities of the new machines often referred to computers "understanding" certain notations or having a "vocabulary."[24] This reflected a conception that the mode of use of these new machines was, in important ways, a matter of communication rather than simple deployment, a theme that itself suggested connections with other aspects of the cybernetic project, such as Claude Shannon's ideas about information.[25]

However, with increasing experience, these pioneers soon recognized that communication between humans and computers might not be completely straightforward. The communication problem came to be framed in linguistic terms, and the notion of translation was naturally invoked as a strategy for bridging the gap. Stibitz described an ideal computing machine as one capable of understanding any mathematical symbolism and

---

ated up through the 1950s. For example, in 1952, Grace M. Hopper could talk in anthropomorphic terms about "educating a computer," but two years later was attempting to remove "the last remaining words of the 'magic brain' class" from a glossary of computing terminology. Two years after that, however, in 1956, programming specialist Saul Gorn still found it natural to use the phrase *electronic brain* in a popular account of computers. See Hopper, "The Education of a Computer"; Hopper et al., *Glossaries of Terms*, 22; and Gorn and Wallace Manheimer, *The Electronic Brain and What It Can Do*.

22. For example, Turing viewed human and mechanical computational agency as being largely interchangeable; see his "On Computable Numbers."

23. Hopper, "The Education of a Computer," 243. For a similarly detailed analogy, see Samuel R. Williams, "Bell Telephone Laboratories' Relay Computing System," particularly the pair of diagrams on p. 58.

24. George R. Stibitz, "The Organization of Large-Scale Computing Machinery," 93, 96, and following pages.

25. Claude E. Shannon, "A Mathematical Theory of Communication" (both articles).

possessing an unlimited vocabulary, which would coincide with that of the mathematician. Aware of the cost and complexity of such a project, however, he was ready to recognize that for a limited range of applications, "the entire burden of *translating* from mathematics to *machine language* may be placed on the operator."[26] There was a long tradition of considering mathematics in linguistic terms—for example, as the language of nature, as the "grammar of science," or, during the twentieth century, as a formal symbolic system. With the anthropomorphic view of automatic calculators at hand, specialists in computer programming quickly extended this linguistic analogy to the expression of mathematical problems on the machines, and to viewing the passage from mathematics to code as one of translation.

By the early 1950s, translation had become a central metaphor used to make sense of the activity of programming (fig. 1). In 1954, for example, the glossary of programming terminology produced by the Association for Computing Machinery (ACM) defined a *compiling routine* as one that "translates a program expressed in pseudo-code into machine code."[27] Similarly, specialists in computer programming started using the phrase *formula translation* to denote attempts to generate machine code automatically from standard mathematical notation.[28] Occasional mentions were made of the possibility of translating between different machine languages, or of developing an order code that would be usable on more than one machine.[29] This tendency was perhaps reinforced by contemporary interest in the machine translation of natural languages, an important field of application of early digital computers that was generously funded by the Department of Defense.[30] In 1954, Margaret Harper, a programmer at

26. Stibitz, "The Organization of Large-Scale Computing Machinery," 94 (emphasis added).

27. ACM, Committee on Nomenclature, *First Glossary of Programming Terminology*, 17. *Translate* was further defined as "to change information (e.g. problem statements in pseudo-code, data, or coding) from one language to another without significantly affecting the meaning" (24). We thank Helena Durnová for showing us this source.

28. Notably in the IBM Corporation's report *Specifications for the IBM Mathematical FORmula TRANslating System FORTRAN*.

29. An informal discussion on "a 'universal' code which could be correctly interpreted not only by programmers but also by various different computers" took place at a conference in December 1951, but concrete steps toward such a code only took place some years later, as described in subsequent sections. See Joint AIEE-IRE Computer Conference, *Review of Electronic Digital Computers*, 113–14.

30. See Hutchins, *Machine Translation*; and Martin-Nielsen, "Private Knowledge, Public Tensions." The development of multilingual machine translation systems was also a main concern for UNESCO at that time, as part of its efforts to preserve the free circulation of scientific knowledge. Machine translation efforts failed in the 1960s, but from this failure came a profound realization about computer understanding of natural languages, informing later research in search engines like Google. We thank one of the reviewers for this insightful remark. On UNESCO's efforts, see, for example, "Working Group to Discuss the Possible Assistance of UNESCO in the Field of Mechanical Trans-

JANUARY
2014
VOL. 55

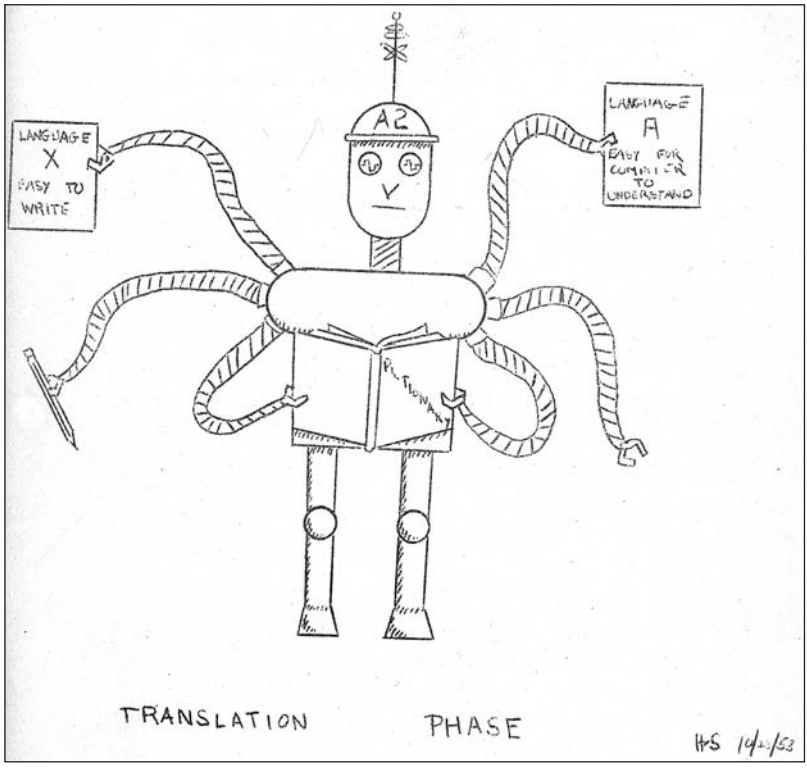


FIG. 1 The first of a series of four robot-like illustrations used by Grace Hopper to explain the A2 compiler for use on the UNIVAC computer during a summer course at the MIT in 1954. (Source: C. W. Adams, ed., *Digital Computers: Advanced Coding Techniques—Notes from a Special Summer Program* [Cambridge, MA: MIT Press: 1954]. Image reproduced by permission of MIT, and courtesy of the Computer History Museum.)

Remington Rand, asked: "If Russian can be translated into English . . . why not one computer code into another?"[31] And a year later, in an internal RAND Corporation document, Allen Newell suggested that through the study of machine translation "[w]e might be a lot closer to 'real' automatic programming of computers."[32]

However, the majority of machines in operation served local commu-

lation," 16 October 1961, available at http://unesdoc.unesco.org/images/0015/001538/153820eb.pdf (accessed 1 November 2012).

31. Margaret H. Harper, "Subroutines."

32. Allen Newell, "Notes," 26 September 1955, and "RAND, Chess Learning Machine Notes," January–September 1955, in Allen Newell Collection, Carnegie Mellon University, Pittsburgh, available at http://doi.library.cmu.edu/10.1184/pmc/newell/box00002/fld00108/bdl0001/doc0003 (accessed 15 November 2013).

nities of scientists and engineers who wanted quick answers to specific calculations, and among these groups of users, there was little practical demand for programs that could run on different machines. Indeed, up until the mid-1950s, coding practices and notations remained highly localized, depending on tools and techniques developed for individual machines. This was the case even for machines with overall design similarities, such as those based on the computer developed by von Neumann's team at the Institute for Advanced Study in Princeton. As machines came into operation and coding became a daily concern, local knowledge and customs, ranging from manual routines to special input and output devices, tended to prevail.

Nevertheless, at several computer installations, these loose groups of users made attempts to share knowledge that transcended local practices. Military funding supported, either directly or indirectly, the majority of computing projects at the time and promoted a relatively open circulation of knowledge, imposing few security restrictions and sponsoring public symposia and the publication of proceedings and newsletters. In addition, a few influential publications describing more general approaches to the programming process circulated widely. In a series of reports published in 1947 and 1948, Herman Goldstine and von Neumann presented a detailed methodology for the design of programs, aiming to give the activity of coding the systematic foundations that would apply across all machine types, and Maurice Wilkes and his collaborators at the University of Cambridge Mathematical Laboratory later published a book that mixed accounts of concrete, local programming practices with the more abstract ideas discussed by von Neumann's group.[33]

In the early 1950s, the emerging field of automatic coding became a locus of shared knowledge about coding and programming techniques. The term *automatic coding* (later also referred to as *autocoding* or *automatic programming*) was variously defined, but was usually taken to refer to the use of the computer itself to take over routine, mechanizable aspects of the programming process, such as the conversion between binary and decimal representations of numbers, the assembly of subroutines taken from a library into complete programs, or the translation of various forms of pseudo-code into machine code.[34] Teams of programmers from many computer facilities developed programs to perform these tasks, such as

33. Herman H. Goldstine and John von Neumann, *Planning and Coding of Problems for an Electronic Computing Instrument*; Maurice V. Wilkes, David J. Wheeler, and Stanley Gill, *The Preparation of Programs for an Electronic Digital Computer*, where in the preface the authors comment that "[t]he methods are described in terms of the code of orders used in the EDSAC, but for the main part they may readily be translated into other order codes."

34. See, for example, the surveys presented by J. H. Brown and John W. Carr III, "Automatic Programming and Its Development on the MIDAC"; and C. W. Adams, "Developments in Programming Research."

JANUARY
2014
VOL. 55

compilers and interpreters, which led to much redundancy and duplication of effort and made clear to the heads of these facilities the need to share information and knowledge.[35]

Automatic coding techniques were, however, highly machine-specific. This was the case even for the most sophisticated of these, the so-called formula translation systems. Based on the observation that some algebraic expressions encode computational procedures, formula translation systems used the computer itself to translate algebraic formulas into equivalent machine code. At MIT, J. Halcombe Laning and Neal Zierler developed the first of these systems for the Whirlwind computer.[36] Their system came to the attention of John Backus, a programmer at IBM who was undertaking the development of an automatic coding system for the forthcoming IBM 704. This new project, known as FORTRAN (for FORmula TRANslation), shared with Laning and Zierler's system the fundamental goal of allowing mathematical notation, in particular algebraic formulas, to be included directly in the programs used by computers.[37]

Formula translation systems thus held out the promise of allowing the universal language of mathematics to be used directly on computers, but that promise was initially only realized for specific machines, whether MIT's Whirlwind or the IBM 704. Before long, however, new events, such as the appearance of multi-machine installations and the advent of the commercial computer, brought about changes in the orientation of programming toward single machines.

New Threats, New Opportunities: The "Commercial Capture of the Computer"

At the beginning of the 1950s, there were a dozen computers in the United States and a handful in the United Kingdom, all of them one-of-a-kind research machines. By 1955, however, there were roughly 200 operational machines worldwide, still mostly in the United States, representing over a hundred different types of machines.[38] One factor in this spectacu-

35. This was the case even with the different sites taking delivery of the same computer—for example, the IBM 701, the first production computer; see Cuthbert C. Hurd, ed., "Special Issue on the IBM 701."

36. In fact, the first concrete realization of this idea came in 1949, with John Mauchly's "short code" for the UNIVAC. Mauchly's proposal required the user to transliterate mathematical symbols into numeric codes, however. See Remington Rand Inc., "Preface"; and J. Halcombe Laning and Neal Zierler, *A Program for Translation of Mathematical Equations for Whirlwind I*.

37. IBM Corporation, *Specifications for the IBM Mathematical FORmula TRANslating System FORTRAN*.

38. The estimates of the total number of computers and their variety are from William Aspray, "International Diffusion of Computer Technology, 1945–1955," 351–52. In Western Europe, by 1955, there were twenty-seven computers installed, fifteen of

lar growth was the appearance of the first production-line machines, such as the UNIVAC and the IBM 701. By the end of 1954, Remington Rand had delivered sixteen UNIVACs, and eighteen 701s had been manufactured and delivered, predominantly to U.S. military and defense contractors, with the balance going to government agencies, insurance companies, and universities.[39] Together, these first commercial machines represented a growth in just a single year of over 20 percent of the total number of operational computers in the country.

This increasing diversity of computers meant that computer users had to adapt to a situation where the use of multiple types of computer was the norm. Some installations soon acquired a range of different machines, whereas others found it desirable to purchase new and more powerful machines as the technology evolved. This situation soon revealed limitations in existing programming techniques, which, as discussed above, were largely machine-specific, meaning that programs coded for machines of one type could not be used on those of another.

This remarkable expansion came at a time of increased work pressure, since many customer installations were involved in defense production work.[40] The lack of machine compatibility could therefore have serious economic consequences; for example, it would significantly raise the costs associated with the purchase of a new machine if all existing programs had to be rewritten for it. A further problem was that the variety of codes used on these machines meant that programming personnel had to be trained to work on a range of different machines; but during a period of manpower shortages in the computing field, this was a luxury few facilities could afford.[41]

Recognizing the urgency of the situation, IBM made extensive efforts during the development of the 701 computer to involve prospective customers in the development of programming expertise and infrastructure.[42]

---

them in the UK. See W. K. Bruijn and Stichting Studiecentrum voor Administratieve Automatisering, *Computers in Europe 1966*, table 1; and Martin H. Weik, *A Survey of Domestic Electronic Digital Computing Systems*.

39. Ceruzzi, *A History of Modern Computing*, chap. 2.

40. From the early 1950s, the increasing replacement of manned aircraft for missiles, as part of a new U.S. military strategy, brought the computing facilities of firms like Douglas and Lockheed under remarkable pressure, since missile design and manufacturing demanded much greater computing accuracy. See Allen J. Scott, "The Aerospace-Electronics Industrial Complex of Southern California"; and G. R. Simonson, "Missiles and Creative Destruction in the American Aircraft Industry, 1956–1961." For an eyewitness account, see Fred J. Gruenberger, "A Short History of Digital Computing in Southern California."

41. Besides, manpower shortages made it difficult to find and retain the best programming personnel and to keep salaries under control; see Ensmenger, *The Computer Boys Take Over*, chap. 3.

42. In part, this move reflected IBM's fear that programming costs would rise to the point whereby companies would have difficulties in justifying the total cost of computing. For an insider's view, see R. Blair Smith, "The IBM 701."

JANUARY
2014
VOL. 55

The strategy was to encourage cooperation among different customers, and between customers and IBM. During 1952–54, IBM, with the RAND Corporation acting as clearinghouse, promoted several projects of collaboration in programming techniques among computing installations in the Los Angeles region.[43] Even these efforts did not suffice to prevent significant duplication of effort, such as the production of different assembly and formula translation programs at various IBM 701 installations.[44]

The problems were discussed by a number of specialists in computer programming in May 1954, during a symposium on automatic coding techniques organized by the Office of Naval Research.[45] In particular, mathematicians Saul Gorn of the U.S. Army's Ballistics Research Laboratory at the Aberdeen Proving Ground in Maryland and John Weber Carr and his colleague J. H. Brown of the Willow Run Research Center at the University of Michigan presented their research and experiments on the possibility of a "universal code" or "universal computer language."[46] Working independently of each other, Gorn and Carr were confronted with similar problems, having to produce large quantities of day-to-day computational work to strict deadlines, while at the same time looking after the training of programming personnel. Certainly, these problems were not unique to Gorn's and Carr's respective computing facilities—quite the opposite. As Nathan Ensmenger has shown, manpower shortages of programming personnel were among the most critical problems in the computing field during the mid-1950s.[47] What made Gorn's and Carr's sit-

43. Atushi Akera, *Calculating a Natural World*, 251–59.

44. See the papers on the different IBM 701 installations collected in Hurd, ed., "Special Issue on the IBM 701."

45. U.S. Navy Mathematical Computing Advisory Panel, ed., *Symposium on Automatic Programming for Digital Computers, 13–14 May 1954.*

46. Saul Gorn, "Planning Universal Semi-Automatic Coding"; Brown and Carr, "Automatic Programming and Its Development on the MIDAC." The possibility of a universal code had been raised before—at an informal discussion at the Joint AIEE–IEE Computer Conference in 1951 (Joint AIEE-IEE Computer Conference, *Review of Electronic Digital Computers*, 113–14), where Charles W. Adams of MIT "suggested the possibility of a 'universal' code which could be correctly interpreted not only by programmers but also by various different computers by means of interpretative subroutines. Such a scheme could permit the same program to be used on different computers, albeit at a loss of efficiency, and might aid in the adoption of a truly universal code for future computers." Little immediate progress toward this goal was made, however, but the 1954 meeting marks a significant quickening of interest in the topic.

47. Indeed, manpower shortages were a problem affecting many technical and scientific fields in the United States at that time, especially those related to the defense effort, such as physics, engineering, and linguistics. For the physics situation, see David Kaiser, "Cold War Requisitions, Scientific Manpower, and the Production of American Physicists after World War II." For contemporary accounts, see Ruth W. Wolfe, "The Technical Manpower Shortage"; Harry H. Ransom, "Scientific Manpower and National Security"; U.S. Congress, Joint Committee on Atomic Energy, *Engineering and Scientific Manpower in the United States, Western Europe and Soviet Russia*; Donald A. Quarles,

uation more urgent is that, although working in different institutional circumstances, they were confronted by the problems caused by a variety of machines and codes on an almost daily basis.

At the Aberdeen Proving Ground, where Gorn was the mathematics advisor to the Computing Lab, the ENIAC, EDVAC, and ORDVAC computers had been in simultaneous use since 1951.[48] This situation prompted him to consider the development of a code simple enough that anybody could learn to use it in a short time, and that could be used to produce programs to run on any machine. Such a code, Gorn believed, would put an end to universities' reluctance to train graduates in programming techniques, as universities often used the existing multiplicity of codes as an excuse not to engage in the training of programming personnel. Gorn's code would allow programmers to describe, in a machine-independent way, the construction of programs out of an agreed set of basic subroutines, and it formed part of a more general and systematic approach to programming that made extensive use of flowcharts. The system was never fully implemented, however, although Gorn was able to produce simple subroutines that were automatically translated by the EDVAC and ORDVAC computers into their own code.

Carr was thinking along similar lines. He was in charge of the MIDAC computer at the Willow Run Research Center, the University of Michigan's off-site military-research facility. Carr's team had to produce high-precision computational work for the U.S. Air Force's BOMARC guided-missile program, in addition to work for the university's departments and other governmental agencies, as well as for industry.[49] Achieving high accuracy was a complex, error-prone, time-consuming programming task, so Carr's prime concern was the development of a mistake-free coding process based on easy-to-correct and easy-to-use input languages and the elimination of as much human intervention as possible.[50] Furthermore, the MIDAC, initially designed exclusively for military use, had been open to Michigan's faculty and students since 1953, becoming a "laboratory tool" to fill the gap caused by a lack of general training on programming.[51] In

---

"Need of Scientific Manpower"; American Mathematical Society and National Research Council, *Proceedings of a Conference on Training in Applied Mathematics*; Robert G. Gibbs, "Manpower Supply Down, Military and Industrial Demand High"; and T. H. Rogers, "Supply and Demand of Technical Personnel in American Industry."

48. Gorn, "Planning Universal Semi-Automatic Coding" and "Standardized Programming Methods and Universal Coding."

49. On John W. Carr III and the computing facility at Willow Run Research Center, see Akera, *Calculating a Natural World*, 295–98, and "The Life and Work of Bernard A. Galler (1928–2006)," 4.

50. Brown and Carr, "Automatic Programming and Its Development on the MIDAC."

51. John W. Carr III and Norman R. Scott, eds., *Notes on Digital Computers and Data Processors*, xii.

JANUARY
2014
VOL. 55

this context, Carr soon realized the advantage of a standard input code. And although, like Gorn, he did not go much further than the development of common automatic coding techniques, he understood very well the benefits of black-boxing the machine: "The progress . . . leads, of necessity, towards some sort of standardization of the basic input language of all computers as it looks to the users, before it is fed through the 'black box' which contains the integrated system which matches the internal language to the external human language."[52]

It was not just about improving human–machine communication; there was also a political side to Carr's views on universality. An ardent defender of the role of universities in computing, he feared that the increasing hegemony of industry in the computer field would endanger the already weak position of the universities.[53] If carried out, such "commercial capture of the computer," as he put it, would leave universities with little control over the new technology.[54] Instead, Carr speculated, the existence of a "common, universal, external language arrived at by mutual agreement and persuasion" would free the user from the manufacturer's machine specifications.[55]

Carr was not alone in raising his voice against the role of industry in the computer field. In February 1954, mathematician Derrick Lehmer of the University of California, Berkeley, who in 1946 had been an early user of ENIAC, expressed similar concerns during his welcoming address to the participants of the second Western Computer Conference in Los Angeles.[56] A member of the computer "old guard," as he introduced himself, Lehmer warned newcomers to the field of the danger that "corporate security" would pose to the free flow of information. Aware of the shift of patronage that was taking place, he feared that industry, about to replace the U.S. government as the driving force in the computer field, would put an end to the "enlightened policy" on the circulation of information implemented by the U.S. government during the previous decade.[57] As Lehmer reminded his audience, government funds had not only supported the creation of computers, they had also sponsored symposia and the publication of proceed-

52. Brown and Carr, "Automatic Programming and Its Development on the MIDAC," 89.

53. John W. Carr III, "Conference Summary," 147.

54. Brown and Carr, "Automatic Programming and Its Development on the MIDAC," 89–90.

55. Ibid.

56. Derrick H. Lehmer, "Welcoming Address." On Lehmer's early programming work, see Maarten Bullynck and Liesbeth de Mol, "Setting-up Early Computer Programs."

57. Lehmer, "Welcoming Address," 8. Indeed, in his speech, Lehmer publicly complained that some signs of this risk had loomed during the preparations of the conference when the program committee decided not to publish transcripts of the discussions following the requests of representatives of private firms.

ings, and encouraged the dissemination of results, as in the case of the Western Computer Conference itself.

As will be shown in the next two sections, the idea of a universal language was to be a decisive step in the emergence of programming languages as distinct objects of knowledge, and eventually of a science of programming. Carr's and Lehmer's warnings may be viewed as boundary work at two levels: first, defending the place of universities within the new commercially oriented computing field; and second, looking for an object of study around which to build a scientific discipline. Indeed, in the summer of 1958, Carr explicitly connected the problem of operating multimachine installations to the opportunity to turn the research area of automatic programming into a scientific discipline:

> If humans are to keep up with the voracious input capacity of the digital computers, then languages built for the humans, and not the machines, must be developed and put into operation. This involves the creation of translators, techniques for using them, and, finally, a theory of such formal translators. Multi-machine installations, with each machine having a separate language of its own, require a unified language for most efficient use. The development of "automatic problem solution" requires formalism, interchangeability of procedures, and computability of languages if it is to become a true discipline in the scientific sense.[58]

Carr's vision, extending into the regions of metalinguistics, derived from his conviction that computer languages were indeed "languages," with their own "simple equivalents of verbs and nouns, tenses and moods."[59]

Despite Lehmer's and Carr's fears, however, the commercial computer was soon to provide computing facilities with the opportunity to increase collaboration on programming. This was largely due to the emergence of user groups—organizations formed by customer firms with the same type of machine that sought to establish working standards to exchange code and co-develop some basic programs, such as assembly and utility subroutines that could be used by all installations.[60] These groups facilitated the transformation of programming into an activity disconnected from the machine, and it was in this context that the idea of common, or universal, languages really caught fire.

58. John W. Carr III, *Computing Programming and Artificial Intelligence*, 2.
59. John W. Carr III, "Methods in High-Speed Computation," 23.
60. Herbert S. Bright, "Computer User Groups."

Varieties of "Universality"

The first formal user group to be established was SHARE, formed in September 1955 by facilities using the IBM 701 and expecting to receive the new IBM 704.[61] Three months later, USE (Univac Scientific Exchange) was established by prospective users of the Remington Rand 1103A computer, including representatives of the computing facilities of Boeing, Holloman Air Force Base, Lockheed Missile Systems Division, Ramo-Wooldridge, and Remington Rand UNIVAC Division. During the following years, inspired by the example of SHARE and USE, many other user groups flourished, with about twenty groups established by 1960.[62]

One of the aims of user groups was to enable the exchange of programs between installations, and to do this, some degree of the standardization of programming tools was clearly required. In the case of USE, this requirement was expressed in terms of language: in December 1955, the first meeting of USE agreed on the adoption of a "common programming language for exchanged programs" as one of the top priorities of the newly established cooperative organization.[63] Conceived as a sort of supra-language to facilitate the exchange of library programs, USE's common language was to include all the languages of specific installations, but without forcing any member to use the common language internally.[64] USE's grand project did not go any further than the development of a standard compiling routine and some other basic programming tools. Nevertheless, it is worth noting that this early use of the term *programming language* appeared in conjunction with the adjective *common*, as if the newly coined phrase sought to convey a sense of detachment from the specific machine.

In the following years, two projects from the user groups and the U.S. military would attempt to broaden the scope of the USE initiative and to achieve a degree of machine independence by developing "universal," "common," or "single" languages that would be usable across computers of different types. The first and less-well-known project was part of the Fieldata Project of the U.S. Army Signal Corps (1956–62), an ultimately unsuccessful though highly ambitious programming effort to develop a "universal computer code," in a similar fashion to SHARE's UNCOL project described below, and aimed to facilitate the portability of computer programs.[65] The second project was the creation of a data-processing language, soon named COBOL (for Common Business-Oriented Language),

61. Atsushi Akera, "Voluntarism and the Fruits of Collaboration."
62. Bright, "Computer User Groups," tables 1–2a, b.
63. Remington Rand Inc., "Formation of USE," 264.
64. Ibid., 265.
65. William F. Luebbert and Percy W. Collom Jr., "Signal Corps Research and Development on Automatic Programming of Digital Computers." Fieldata Project materials can be found in the U.S. Government Computing Collection, Charles Babbage Institute, University of Minnesota, Minneapolis (hereafter CBI-UM).

sponsored and supported by the Department of Defense to reduce the costs associated with the maintenance of a growing variety of computer systems.[66]

Specialists in computer programming had taken the first steps toward machine independence in the area of formula translation languages. Starting in late 1955, a "mathematical language compiler" had been conceived for the Datatron 205 computer at Purdue University.[67] Alan Perlis, a member of the group, then moved to Carnegie Mellon, where he further implemented the language—by then known as IT—for the IBM 650.[68] Similarly, by 1958, IBM was planning to make FORTRAN available on machines other than the IBM 704.[69] Inspired by Perlis's IT, IBM developed the FOR TRANSIT system to enable a limited version of FORTRAN to be translated into IT, thus making it available on the 650.[70] This greatly increased the reach of FORTRAN, hitherto available only on IBM's expensive high-end scientific machines.

These experiments highlighted the promise of the idea of a machine-independent language, but also its limitations: the two versions of IT differed in certain ways, and it proved difficult to implement a full version of FORTRAN for the 650 because of the machine's limited capabilities. However, it began to seem plausible that a form of universality could be achieved by means of translation among languages. As Robert Bemer of IBM put it later in the year, "[a]lthough the ultimate in language does not exist yet, we can console ourselves meanwhile with compatible (as against common) language. There is much current evidence that existing algebraic languages are all mappable into one another by pre-processors."[71]

Institutional interest in the possibility of common languages came initially from the user groups. By this point, user groups had clearly understood that their cooperative efforts were limited to each group's scope; at the same time, however, many member facilities were already operating several types of computers. The problem of information exchange had thus become an in-house issue. So user groups were open to exploring the possibility of a national organization that would transcend the existing user

66. J. A. N. Lee, ed., "Special issue, COBOL: 25th Anniversary"; and Jean E. Sammet, "The Early History of COBOL."

67. The IT language was first described in J. Chipps et al., "A Mathematical Lan-guage Compiler"; and Alan J. Perlis and J. W. Smith, "A Mathematical Language Compiler."

68. Alan J. Perlis, "Two Thousand Words and Two Thousand Ideas." In 1957, IT was released as a library program for the IBM 650, at which point it became known as Internal Translator (IT).

69. FORTRAN was implemented for the IBM 650 and 709 computers in 1958; see J. A. N. Lee, "Pioneer Day, 1982," 10–11. By mid-1958, IT was available on the Burroughs 205 (Datatron) computer. For IT, see Sylvia Orgel, *Purdue Compiler General Description*.

70. IBM Corporation, *Reference Manual FOR TRANSIT Automatic Coding System*. See also David Hemmes, "FORTRANSIT Recollections."

71. Robert W. Bemer, "The Status of Automatic Programming for Scientific Problems," 115.

JANUARY
2014
VOL. 55

groups and include all users of computers, whoever their manufacturer. More important, they were also ready to discuss whether plans should be made for a "truly universal programming language."[72]

Following an initial call from Frank Engel (Westinghouse/SHARE) and Walter Bauer (Ramo-Wooldridge/ACM), the Los Angeles Meeting on Information Exchange assembled representatives of the user groups SHARE, USE, and DUO (Datatron User's Organization), under informal mediation of the ACM, at Ramo-Wooldridge in Los Angeles during 9–10 May 1957 to examine the "ways and means of facilitating the exchange of all types of computing information."[73] This meeting brought together the heads of eight computing installations from major defense contractors, universities, and the federal government, who also attended as representatives of the user groups and the ACM.[74]

During the first day, it soon became clear that the idea of a super-organization would have little endorsement. Although the ACM appeared to be the foremost candidate for such a role, given its independence from industry and its programming-oriented character, the lack of funds and drive of the organization were considered major shortcomings by the user groups. The groups' representatives at Los Angeles were, however, more positive about the idea of a "single universal computer language."[75] This idea was based on evidence, such as the work on FORTRAN and IT described above, that programming techniques were about to transcend the characteristics of particular machines to produce "programs which are machine independent."[76] Still, the participants conceded, such objectives would probably evolve from several initial "universal" languages. The meeting ended with the formulation of a general recommendation to the ACM to "appoint a committee to study and recommend action toward a universal

72. Letter, Walter F. Bauer to Paul Armer, 12 April 1957, 2, series 3, box 7, folder "SHARE Correspondence 1957," Paul Armer Collection, in Archives Center, National Museum of American History, Smithsonian Institution, Washington, D.C. (hereafter AC-NMAH).

73. "Recapitulation of the May, 1957, Los Angeles Meetings on Information Exchange," 1, Universal Language Committee, UNCOL, 8 April 1958, appendix A, box 1, folder 13, SHARE, Inc. Records, in CBI-UM.

74. See ibid.; the attendees were: Paul Armer (RAND/SHARE), Walter F. Bauer (Ramo-Wooldridge/ACM), John W. Carr III (University of Michigan/president of the ACM), Frank Engel (Westinghouse/SHARE chairman), Donald W. Gantner (Ramo-Wooldridge/USE), Edward M. McCormick (U.S. Naval Ordnance Labs/DUO chairman), Robert P. Rich (Johns Hopkins University/USE chairman), and Richard B. Talmadge (Lockheed Aircraft/USE ex-chairman). Unable to attend were Alan J. Perlis (Carnegie Institute of Technology), Jack Strong (North American Aviation), Randall Porter (Boeing), and Walter Ramshaw (United Aircraft).

75. See ibid., 3.

76. Paul Armer et al., "Los Angeles, May 10, 1957," Universal Language Committee, UNCOL, SHARE, Inc. Records, in CBI-UM. Perlis's IT was explicitly mentioned in the letter of invitation of Bauer to Armer, Paul Armer Collection, in AC-NMAH. Perlis himself, unable to attend the meeting, endorsed the final resolution.

programming language," plus three other actions to facilitate the rapid exchange of practical information on programming.[77]

The Los Angeles meeting was to have further significant consequences for the future development of computer programming. One of its first results was the establishment by the ACM of the monthly journal *Communications of the ACM*. At the meeting, Carr, who attended as the president of ACM, had suggested the possibility of a "problem solver's journal" edited by the ACM as a tool for the rapid dissemination of information, with a special stress on programming techniques.[78] Under the editorial leadership of Perlis, the first issue appeared in January 1958, subsequently becoming a worldwide reference source for the publication of research on programming.[79] The second result was that the meeting would eventually lead to the establishment, in January 1958, of an ACM Ad Hoc Committee on Languages. As described below, this committee, in collaboration with a group of heads of computing centers in Europe, was to draw up the first definition of the programming language ALGOL—an ambitious attempt to create a universal language for scientific computation.[80]

The Los Angeles meeting also turned out to be a catalyst for an alternative approach to the question of universality about to gather momentum within SHARE. In the spring of 1957, the so-called Project for the Advancement of Coding Techniques (PACT) came to an end. A forerunner of SHARE, PACT had started in November 1954 as a joint venture of several defense contractors on the West Coast to cooperate with IBM in the development of a more elaborated coding system for the 701 and 704 computers than IBM's FORTRAN.[81] However, the project suffered many delays and never received much use; the second version, conceived for the 704, remained unfinished. As an alternative course of action, the final report of the PACT II Working Committee, released in the spring of 1957, recommended the "design of a universal, intermediate language, independent of specific hardware but similar in character to machine languages." The UNCOL project was born.[82]

In the winter of 1957, SHARE established an Ad Hoc Committee on

77. Armer et al., "Los Angeles, May 10, 1957," SHARE, Inc. Records, in CBI-UM.

78. See "Recapitulation of the May, 1957, Los Angeles Meetings on Information Exchange," 2, SHARE, Inc. Records, in CBI-UM.

79. In the fall of 1957, mathematician Franz L. Alt of the National Bureau of Standards, together with Carr, Bauer, and Perlis, among others, would give the main impetus to the establishment of the *Communications of the ACM*; see Walter F. Bauer, Mario L. Juncosa, and Alan J. Perlis, "ACM Publication Policies and Plans," 121–22.

80. For more details on these events, see David Nofre, "Unraveling Algol," 62–64.

81. Initially (November–December 1954), the project included the following West Coast computing installations: Douglas Aircraft (El Segundo and Santa Monica), IBM, North American Aviation, Ramo-Wooldridge, and RAND Corporation; see "PACT Notes," 1 December 1954, box 22, folder 7, John Clifford Shaw Papers, in AC-NMAH.

82. T. B. Steel Jr., chairman, UNCOL Committee, to SHARE, "Committee Report," 17 March 1961, 2, box 1, folder 32, Francis V. Wagner Papers, in CBI-UM.

JANUARY
2014
VOL. 55

Universal Languages to further develop the original recommendations of PACT II's final report. The ACM's new journal published this committee's report in the summer of 1958.[83] The committee distinguished between "problem-oriented languages" (POLs), such as FORTRAN and IT and equivalent languages for tasks like data processing, and machine languages (MLs).[84] Its conclusions were based on a particular view about universality—namely, that "[i]t is impossible to agree on one universal POL."[85] Because of the ever-increasing range of tasks that computers were being used to solve, SHARE's Ad Hoc Committee on Universal Languages felt that a universal problem-oriented language would either be inadequate for some areas or else unwieldy and unusable. But in the absence of a universal language, a compiler would have to be written for every required combination of problem-oriented language and machine.

To avoid this, the committee introduced an intermediate linguistic level between the problem-oriented languages and the machines, consisting of a single Universal Computer Oriented Language (UNCOL). It was planned that UNCOL would be a machine-like language that abstracted from the idiosyncrasies of specific machines; the idea was to produce an UNCOL-to-machine-language translator for each given machine, along with generators to translate each required problem-oriented language into UNCOL. The SHARE committee argued that this would be more economically feasible than writing a compiler for each problem-oriented language[86] (fig. 2).

In rejecting the idea of a universal problem-oriented language and locating the idea of universality closer to the machine, the UNCOL project separated the exchange of computer code, expected to be carried out at the intermediate level, from the exchange of programming ideas and techniques as embodied in human-readable, problem-oriented languages. Yet, it turned out to be much harder than expected to create a computer-oriented language that was independent of the wide range of contemporary machine architectures. By 1961, after three years of intense work, SHARE's UNCOL committee, a descendant of the original committee, had only been able to put together a first version of a proposal; one year later, the project ceased to exist.[87]

83. J. Strong et al., "The Problem of Programming Communication with Changing Machines."

84. In the UNCOL project, the term *problem-oriented language* referred to a "language most natural to [an individual's] way of thinking about the problem" (ibid., 12). In the early 1960s, this term was sometimes regarded as interchangeable with *procedure-oriented language*, but the latter phrase gradually acquired the distinct meaning of a language "in which the user specifies a set of executable operations which are to be performed in sequence"; see Jean E. Sammet, *Programming Languages*, 19.

85. Strong et al., "The Problem of Programming Communication with Changing Machines," 13.

86. For the economic argument, see T. B. Steel, "UNCOL."

87. The idea of a universal intermediate language has been revived on different occa-

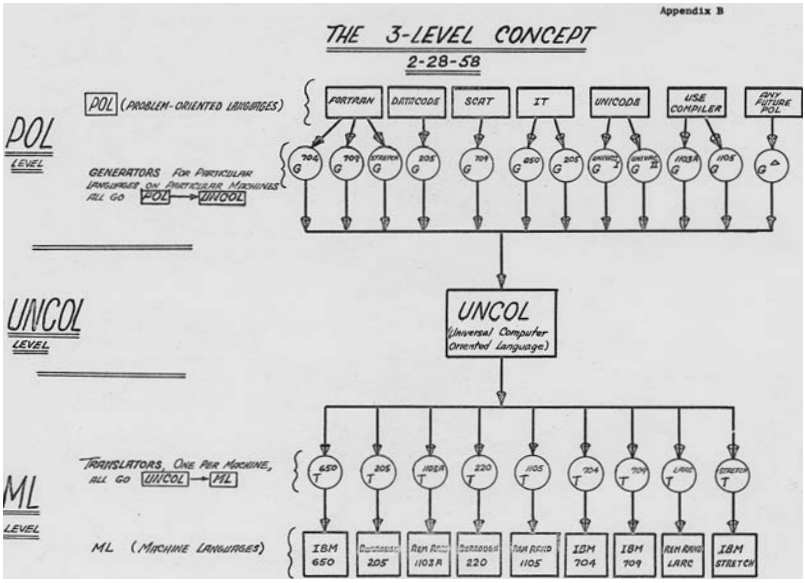


FIG. 2 A schematic showing the UNCOL three-level hierarchy of language, 28 February 1958. (Source: Appendix B of the report "Recapitulation of the May, 1957, Los Angeles Meeting on Information Exchange," Francis V. Wagner papers (CBI 6), box 1, folder 28. Image reproduced courtesy of the Charles Babbage Institute, University of Minnesota, Minneapolis.)

In today's technical understanding, it may seem that problem-oriented languages—in effect, high-level programming languages—were, in fact, the solution to the portability problem; however, the overall situation at this time was rather confusing. The understanding of programming as a process of human–machine communication was starting to falter, and the notion of language was proving to be fruitful in the development of a machine-independent approach to computer programming. But the move to a machine-independent notion of language was not immediate: programming languages were still thought of as parts of programming systems tied to specific machines. Not surprisingly, some felt that the "definition of language was like the proverbial bucket of worms."[88] It was in connection with the ALGOL project that the contours of a broader notion of language emerged.

---

sions, most recently by the programming language Java and its virtual machine. We thank one of the anonymous reviewers for this observation.

88. "Panel Discussion on Universal Language," 8, *Proceedings of the 10th Meeting of SHARE, 26–28 February 1958, Washington, D.C.*, box 3, folder 16, appendix E, SHARE, Inc. Records, in CBI-UM.

Universality Goes Worldwide

The establishment of the ACM Ad Hoc Committee on Languages was a result of the Los Angeles meeting between user groups and the ACM in the spring of 1957. But during the following months, the discussions between the user groups did not bring about any concrete results. Meanwhile, representatives of a West German–Swiss alliance of academic computing installations, working on a common formula-translation project under the sponsorship of the German Society for Applied Mathematics and Mechanics (GAMM), contacted Carr, the president of the ACM, to propose a joint conference to establish a "common formula language."[89] The European initiative came at the right time for Carr and the ACM, both then struggling in their negotiations with the user groups. But it came at a price, for it added an international dimension to the already wide range of meanings of universality that were in play.

The ACM Committee on Languages first met during 24–25 January 1958 at the Carnegie Institute of Technology under the chairmanship of Perlis to discuss how to proceed further in relation to the West German–Swiss proposal.[90] Two weeks later, John Backus prepared a memorandum based on what the committee had so far agreed on, which set forth the purposes and basic properties of the "proposed programming language":

> A. To provide a uniform internationally accepted language in which to publish procedures for the solution of a variety of "scientific" problems, a language which is at once precise, concise and easily understood by those familiar with the few simple rules needed for its interpretation.
>
> B. To provide a programming language which, on the one hand, is sufficiently powerful and concise to compare favorably with the best such languages known today and which, on the other hand, appears feasible and likely to be adopted as the input language for a large class of computers both in the United States and in Europe.[91]

89. On the details of these contacts, see Nofre, "Unraveling Algol," 63. The quote is from the English translation of the official letter of invitation: Heinz Rutishauser et al. to John W. Carr III, president of ACM, 19 October 1957, enclosed in the memorandum from Carr to the officers and council of the ACM, 13 December 1957, which is part of Universal Language Committee, UNCOL, SHARE, Inc. Records, in CBI-UM.

90. "News and Notices," 14. The membership of the committee was: J. W. Backus (IBM), P. H. Desilets (Remington Rand), D. C. Evans (Bendix Aviation), R. Goodman (Westinghouse), H. Huskey (UC, Santa Cruz), C. Katz (Remington Rand), J. McCarthy (MIT), A. Orden (Burroughs), A. J. Perlis (Carnegie Institute of Technology), R. Rich (Johns Hopkins University), S. Rosen (Burroughs), W. Turanski (Remington Rand), and J. Wegstein (National Bureau of Standards). See "Proposal for a Programming Language" (n.d. [ca.May 1958]), Unprocessed Computer Documentation Collection, in AC-NMAH.

91. Memorandum, J. Backus to the ACM Ad Hoc Committee on Languages, 12 February 1958, box 264, Charles Katz Papers, in AC-NMAH.

This distinction between *language* and *programming language* makes it clear that the ACM committee considered that computer programs were not merely devices to get a machine to do a particular computation, but that they encapsulated potentially valuable knowledge that, as well as being shared on a concrete level, was worthy of a more traditional form of scientific publication. But existing languages did not meet the purpose. For instance, formula translation systems allowed some mathematical notation, in particular algebraic expressions, to be used in programs; yet, their control statements, which defined the overall structure of the computation, were highly diverse and machine-specific. If for no other reason, a new language was necessary to provide a universally accepted notation for control. Therefore, the aim was not simply to define a new programming language, but to provide new standards in mathematical notation capable of describing the structure of extended computations, as well as their individual steps.[92]

A few months later, a meeting between representatives of the ACM committee and the West German–Swiss group took place in Zurich between 27 May and 2 June. Four members from each side attended the meeting; the American delegation included both Backus and Perlis, principal designers, respectively, of the FORTRAN and IT languages. What emerged from this meeting was a new kind of language, dubbed an "algorithmic language." Rather than being a vehicle for coding computers, the new International Algebraic Language (IAL), defined and published in the form of a scientific article, was thought of primarily as a kind of scientific notation, an extension of mathematical language.[93] Echoing Backus's earlier memorandum, the report stated that "[t]he purpose of the algorithmic language is to describe computational processes";[94] it was further stipulated that the language should be "as close as possible to standard mathematical notation," and that "it should be possible to use it for the description of computing processes in publication."[95]

The final objective stated that IAL should be "mechanically translatable into machine programs."[96] However, the proposal was not written with the properties of any one machine in mind, and certain areas of functionality, such as the definition of methods for input and output, were omitted. This was partly because these would be hard to specify in a universally applica-

92. As Robert Bemer expressed it in October 1957, "[t]he area of loop control and recursive operations is still not well handled in existing mathematical notation, but computers are forcing the development"; see Bemer, "The Status of Automatic Programming for Scientific Problems," 115.

93. Alan J. Perlis and Klaus Samelson, "Preliminary Report." The report was also published in the first issue of the journal of numerical analysis, *Numerische Mathematik*, established with the collaboration of some of the members of the Western German–Swiss group, such as Friederich L. Bauer and Heinz Rutishauser.

94. Ibid., 10.

95. Ibid., 9.

96. Ibid.

JANUARY
2014
VOL. 55

ble way; a more significant reason was that such details did not form part of the "scientific" aims of the language—the definition of computational processes. Besides, the existing state of computing hardware made the translation of a universal language somewhat problematic, as computers used widely varying input devices, often with very limited character sets.[97] The committee therefore defined three distinct linguistic levels: the central level was the so-called reference language, based on a stipulated character set adopted for the purposes of defining the language.[98] This could be interpreted in a number of hardware representations to take into account the peculiarities of individual machines, and could also be presented in one or more publication languages. These would enable the use of a greater range of mathematical notations, and also permit national and other variations, such as the symbol used for the decimal point. Although these distinctions seem trivial now, they marked a growing awareness that a language could be thought of as not just a set of physical inscriptions, but instead as an abstract structure whose properties could be formally defined and studied.

IAL aimed at multiple levels of universality. Aspiring to be an international and universal human language, analogous to that of mathematics, it combined this goal with the apparently conflicting ambition to be usable on a wide range of computing equipment by reifying the metaphor of "computer code as language" into the definition of an abstract structure that could be concretely manifested in as many different ways as necessary.

Shortly after the publication of the report, a number of groups began to write translators for IAL, soon renamed ALGOL (for ALGOrithmic Language). Ironically enough, different implementers made independent and incompatible choices based on local perceptions and needs. As IAL/ALGOL, unlike earlier proposals like FORTRAN, was explicitly called, and treated as, a language, its variants naturally came to be referred to as "dialects." At the same time, the report provided an immediate focus for debate on the very idea of a programming language. Readers of the *Communications of the ACM* and the newly formed *ALGOL Bulletin* commented on and criticized the detailed proposals, and more general discussions took place at the International Conference on Information Processing, held in Paris under the auspices of UNESCO, in June 1959. As a result of these discussions, a second committee met in Paris in January 1960 to reconsider the language, consequently defining a new language, called ALGOL 60.[99]

97. The Datatron, for which IT was initially planned, could only accept input in the form of decimal digits, and the language was defined using what was, in effect, a reference language; see Chipps et al., "A Mathematical Language Compiler." In the IAL report, however, the abstract notation rather than the hardware character set was, for the first time, treated as fundamental.

98. Perlis and Samelson, "Preliminary Report."

99. Peter Naur et al., "Report on the Algorithmic Language ALGOL 60." For the use of the term *dialect*, see, for example, Harry D. Huskey, "NELIAC—a Dialect of ALGOL."

One of the most significant advances of this new proposal was that ALGOL 60 was defined completely formally, using a notation presented by Backus in 1959 involving a *syntax* and *semantics* in a manner reminiscent of Rudolf Carnap's model of a formal language.[100] This made clear that ALGOL 60 itself was a formal mathematical, linguistic object, not just a messy adjunct to the technical business of getting a computer to run. As the introduction of the metaphors of syntax and semantics reveals, Backus's proposal emerged from a growing engagement among the fields of computing, formal logic, and linguistics. In this regard, the ALGOL 60 report proved to be a major catalyst in the application of results and techniques from these fields to the area of computer programming, establishing among other things the study of algorithms and programming languages themselves as a central part of the emerging discipline of computer science.[101]

Conclusion

Matthias Dörries has convincingly argued that scientists consistently push metaphors to the limit, exploring their potential both as a means of thinking and as explanatory tools.[102] This was also the case for the language metaphor in computer programming. This article has outlined some key aspects of the genealogy of the term *programming language*. We began by noting that the use of automatic digital computers was, from the beginning, understood as a communicative relationship between two autonomous entities rather than as a passive relationship of deployment. We showed how this relationship manifested itself in specifically linguistic terms with, at its heart, the notion of translation between human and machine languages.

This simple model began to break down as the number and variety of computers grew, leading to calls for the creation of common languages that could mediate the growing "confusion of tongues" and enable the economic migration of computer code from machine to machine, both within and among organizations. At the same time, a small community of computer specialists in academic computer centers worked to establish a space within which the free exchange of information and collaboration charac-

100. J. W. Backus, "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference." For a detailed discussion of the influence of logic in ALGOL, see Mark Priestley, *A Science of Operations*, chap. 8.

101. The importance of the algorithm as a central concept for an emerging computer science is well-described by Ensmenger, *The Computer Boys Take Over*, chap. 5; the importance of ALGOL 60 to the institutional foundation of the discipline is in Priestley, *A Science of Operations*, chap. 9. One immediate consequence was an upsurge of interest in formal language theory and its application to compilation. As the UNCOL project had noted, compilation was a significant reverse salient in programming language technology, and the ALGOL 60 report inspired a great effort to remove this obstacle.

102. Dörries, ed., "Language as a Tool in the Sciences," 4.

JANUARY
2014
VOL. 55

teristic of a period of military funding could continue as computer development increasingly became the preserve of large commercial companies, notably IBM. As a result, two prominent senses of the notion of *universality* emerged: first, the idea of machine independence so that research and development in programming would not be constrained by the demands of particular manufacturers; and second, a sense connecting directly with the universality of the notations of science—and in particular, mathematical notation.[103]

The notion of a programming language, which is connected to the idea of universality, became central to this exercise of boundary work that sought to disengage the activity of programming from local conventions, and to transform it into a transcendent and universal body of knowledge. From this endeavor, programming languages and algorithms emerged as epistemic objects stripped of any marks that would associate them with specific hardware. In the 1960s, as Brent Jesiek has noted, programming languages and algorithms would gain ever more relevance in the design of academic curricula as part of computer scientists' efforts to insulate their nascent discipline from rapid technological development.[104] To what point this strategy succeeded is a question that deserves further discussion. The rise of software engineering in the early 1970s and the existence of a growing body of programmers without any professional training are just two reminders of the limitations of this agenda. Nonetheless, the linguistic conception of computer programming was here to stay, providing a specific vocabulary, new categories of problems and even scientific paradigms for the emerging field of computer science, and, most of all, acting as an "island of semantic stability" in a rapidly changing technological environment.[105]

More work needs to be done in detailing how programming languages were consolidated by basing them on the existing notion of a formal language developed by logicians in the 1930s and appropriated by linguists during the postwar years. In particular, machine-translation projects deserve further attention as spaces of interaction for the communities of linguists, logicians, and programming specialists. In any case, by the early 1960s, the connection between programming languages and formal languages came to be fully recognized. And programming languages and the programs written in them were about to acquire a new status as objects of knowledge in their own right: from being a question of communicating with a machine, programming had become a linguistic activity—the writing of programs.

103. In connection with the first point, Bemer, then at IBM, commented in 1957 that computer manufacturers should provide not only the machine, but also, "with the full co-operation of responsible users," the means of using it—namely, the programming languages; see Bemer, "The Status of Automatic Programming for Scientific Problems," 117.

104. Brent K. Jesiek, "Between Discipline and Profession," 177.

105. On the notion of an "island of semantic stability," see Pierre Laszlo, "Conventionalities in Formula Writing," 52.

Bibliography

Archival Sources

Allen Newell Collection, Carnegie Mellon University, Pittsburgh

Archives Center, National Museum of American History, Smithsonian Institution, Washington, D.C.
Charles Katz Papers
John Clifford Shaw Papers
Paul Armer Collection
Unprocessed Computer Documentation Collection

Charles Babbage Institute, University of Minnesota, Minneapolis
Association for Computing Machinery (ACM) Exhibit
Francis V. Wagner Papers
SHARE, Inc. Records

Published Sources

Adams, C. W., ed. *Digital Computers: Advanced Coding Techniques—Notes from a Special Summer Program*. Cambridge, MA: MIT Press, 1954.

_____. "Developments in Programming Research." In *Proceedings of the Eastern Joint Computer Conference: Papers and Discussions Presented at the Joint ACM–AIEE–IRE Computer Conference Boston, Mass., November 7–9, 1955*, edited by Joint ACM–AIEE–IRE Computer Conference, 75–78. New York: IRE, 1956.

Akera, Atsushi. "Voluntarism and the Fruits of Collaboration: The IBM User Group, Share." *Technology and Culture* 42, no. 4 (2001): 710–36.

_____. *Calculating a Natural World: Scientists, Engineers, and Computers during the Rise of U.S. Cold War Research*. Cambridge, MA: MIT Press, 2007.

_____. "The Life and Work of Bernard A. Galler (1928–2006)." *IEEE Annals of the History of Computing* 30, no. 1 (2008): 4–14.

American Mathematical Society and National Research Council. *Proceedings of a Conference on Training in Applied Mathematics, Columbia University, New York City, 22–24 October 1953*. New York: National Academies, 1953.

Aspray, William. "From Mathematical Constructivity to Computer Science: Alan Turing, John von Neumann, and the Origins of Computer Science in Modern Logic" (Ph.D. diss., University of Wisconsin–Madison, 1980).

_____. "International Diffusion of Computer Technology, 1945–1955." *IEEE Annals of the History of Computing* 8, no. 4 (1986): 351–60.

Association for Computing Machinery (ACM). *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*. New York: ACM, 2007.

_____, Committee on Nomenclature. *First Glossary of Programming Terminology: Report to the Association for Computing Machinery*, June

JANUARY
2014
VOL. 55

1954, box 1, folder 30, Association for Computing Machinery (ACM) Exhibit, in Charles Babbage Institute, University of Minnesota, Minneapolis.

Babbage, Charles. "On the Mathematical Powers of the Calculating Engine." In *The Works of Charles Babbage*, vol. 3, edited by Martin Campbell-Kelly, 15–61. London: Pickering, 1989.

Backus, J. W. "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference." In *Information Processing: Proceedings of the International Conference on Information Processing, Unesco, Paris, 15–20 June 1959*, edited by UNESCO, 125–31. Paris: UNESCO, 1960.

Bauer, Friedrich L., and Klaus Samelson. "The Problem of a Common Language, Especially for Scientific Numeral Work." In *Information Processing: Proceedings of the International Conference on Information Processing, Unesco, Paris, 15–20 June 1959*, edited by UNESCO, 120–24. Paris: UNESCO, 1960.

Bauer, Walter F., Mario L. Juncosa, and Alan J. Perlis. "ACM Publication Policies and Plans." *Journal of the ACM* 6, no. 2 (1959): 121–22.

Bemer, Robert W. "The Status of Automatic Programming for Scientific Problems." In *Proceedings of the Fourth Annual Computer Applications Symposium, October 24–25, 1957*, edited by Francis C. Bock, 107–17. Chicago: Armour Research Foundation, 1958.

Bensaude-Vincent, Bernadette. "Languages in Chemistry." In *The Cambridge History of Science: The Modern Physical and Mathematical Sciences*, edited by Mary Jo Nye, 174–90. Cambridge: Cambridge University Press, 2003.

Bergin, Thomas J., and Richard G. Gibson Jr. *History of Programming Languages II*. New York: ACM Press, 1996.

Berkeley, Edmund C. *Giant Brains, or Machines That Think*. New York: John Wiley and Sons, 1949.

Blackwell, Alan F. "The Reification of Metaphor as a Design Tool." *ACM Transactions on Computer–Human Interaction* 13, no. 4 (2006): 490–530.

Bowden, B. V., ed. *Faster Than Thought: A Symposium on Digital Computing Machines*. New York: Pitman, 1953.

Bowles, Mark D. "U.S. Technological Enthusiasm and British Technological Skepticism in the Age of the Analog Brain." *IEEE Annals of the History of Computing* 18, no. 4 (1996): 5–15.

Bright, Herbert S. "Computer User Groups." Unpublished document, 1960. (Reprinted in *IEEE Annals of the History of Computing* 12, no. 1 [1990]: 56–61.)

Brown, J. H., and John W. Carr III. "Automatic Programming and Its Development on the MIDAC." In *Symposium on Automatic Programming for Digital Computers, 13–14 May 1954,* edited by U.S. Navy

Mathematical Computing Advisory Panel, 84–97. Washington, DC: U.S. Department of Commerce/Office of Technical Services, 1954.

Bruijn, W. K., and Stichting Studiecentrum voor Administratieve Automatisering. *Computers in Europe 1966: A Progress Report on the Development of the European Computer Market*. Amsterdam: Automatic Information Processing Research Centre, 1966.

Bullynck, Maarten, and Liesbeth de Mol. "Setting-up Early Computer Programs: D. H. Lehmer's ENIAC Computation." *Archive for Mathematical Logic* 49, no. 2 (2010): 123–46.

Burks, Arthur W., Herman H. Goldstine, and John von Neumann. *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. Princeton, NJ: Institute for Advanced Study, 28 June 1946.

Campbell-Kelly, Martin. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. Cambridge, MA: MIT Press, 2004.

_____, and William Aspray. *Computer: A History of the Information Machine*. New York: Basic Books, 1996.

Carr, John Weber, III. "Methods in High-Speed Computation." In *Notes on Digital Computers and Data Processors: Prepared for Special Summer Conference at the University of Michigan*, edited by John Weber Carr III and Norman R. Scott, III.5.2.1–38. Ann Arbor: University of Michigan Press, 1955.

_____. "Conference Summary." In *Proceedings of the Eastern Joint Computer Conference, 10–12 December 1956*, edited by Joint Computer Conference, 147–50. New York: AIEE, 1957.

_____. *Computing Programming and Artificial Intelligence: An Intensive Course for Practicing Scientists and Engineers—Lectures Given at the University of Michigan, Summer 1958*. Ann Arbor: University of Michigan Press, 1958.

_____, and Norman R. Scott, eds. *Notes on Digital Computers and Data Processors: Prepared for Special Summer Conference at the University of Michigan*. Ann Arbor: University of Michigan Press, 1955.

Ceruzzi, Paul E. *A History of Modern Computing*. Cambridge, MA: MIT Press, 1998.

Chipps, J., M. Koschmann, S. Orgel, A. Perlis, and J. Smith. "A Mathematical Language Compiler." In *Association for Computing Machinery Eleventh Annual Meeting, UCLA, 27–29 August 1956*, edited by ACM, 114–17. New York: ACM Press, 1956.

Chun, Wendy Hui Kyong. *Programmed Visions: Software and Memory*. Cambridge, MA: MIT Press, 2011.

Colburn, Timothy R., and Gary M. Shute. "Metaphor in Computer Science." *Journal of Applied Logic* 6, no. 4 (2008): 526–33.

Crowther-Heyck, Hunter. "George A. Miller, Language, and the Computer Metaphor of Mind." *History of Psychology* 2, no. 1 (1999): 37–64.

Dörries, Matthias, ed. "Language as a Tool in the Sciences." In *Experi-*

JANUARY
2014
VOL. 55

menting with Tongues: Studies in Science and Language, 1–20. Palo Alto, CA: Stanford University Press, 2002.

Edwards, Paul N. *The Closed World: Computers and the Politics of Discourse in Cold War America*. Cambridge, MA: MIT Press, 1996.

Ensmenger, Nathan. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. Cambridge, MA: MIT Press, 2010.

Fuller, Matthew. *Software Studies: A Lexicon*. Cambridge, MA: MIT Press, 2008.

Galison, Peter. "The Ontology of the Enemy: Norbert Wiener and the Cybernetics Vision." *Critical Inquiry* 21, no. 1 (1994): 228–66.

Gibbs, Robert G. "Manpower Supply Down, Military and Industrial Demand High." *Industrial and Engineering Chemistry* 46, no. 6 (1954): 1139–44.

Goldstine, Herman H., and John von Neumann. *Planning and Coding of Problems for an Electronic Computing Instrument: Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument*. Princeton, NJ: Institute for Advanced Study, 1947.

Gorn, Saul. "Planning Universal Semi-Automatic Coding." In *Symposium on Automatic Programming for Digital Computers, 13–14 May 1954*, edited by U.S. Navy Mathematical Computing Advisory Panel, 74–83. Washington, DC: U.S. Department of Commerce/Office of Technical Services, 1954.

_____. "Standardized Programming Methods and Universal Coding." *Journal of the ACM* 4, no. 3 (1957): 254–73.

_____, and Wallace Manheimer. *The Electronic Brain and What It Can Do*. Chicago: Science Research Associates, 1956.

Gruenberger, Fred J. "A Short History of Digital Computing in Southern California." *Computing News* 7, no. 145 (1959). (Reprinted in *IEEE Annals of the History of Computing* 2, no. 3 [1980]: 246–50.)

Haigh, Thomas. "Software in the 1960s as Concept, Service, and Product." *IEEE Annals of the History of Computing* 24, no. 1 (2002): 5–13.

Harper, Margaret H. "Subroutines: Prefabricated Blocks for Building." *Computers and Automation* 3, no. 3 (1954): 14–15.

Harris, Randy Allen. "Chomsky's other Revolution." In *Chomskyan (R)evolutions*, edited by Douglas A. Kibbee, 238–64. Amsterdam: John Benjamins Publishing Company, 2010.

Hashagen, Ulf, Reinhard Keil-Slawik, and Arthur L. Norberg. *History of Computing: Software Issues*. New York: Springer, 2002.

Hemmes, David. "FORTRANSIT Recollections." *IEEE Annals of the History of Computing* 8, no. 1 (1986): 70–73.

Holmevik, Jan Rune. *Inside Innovation: The Simula Research Laboratory and the History of the Simula Programming Language*. Oslo: Simula Research Laboratory, 2004.

Hopper, Grace M. "The Education of a Computer." In *Proceedings of the 1952 ACM National Meeting (Pittsburgh)*, edited by ACM and C. V. L. Smith, 243–49. New York: ACM Press, 1952.

_____, et al. "Glossaries of Terms—More Discussion." *Computers and Automation* 3, no. 3 (1954): 21–24.

Hurd, Cuthbert C., ed. "Special Issue on the IBM 701." *IEEE Annals of the History of Computing* 5, no. 2 (1983).

Huskey, Harry D. "NELIAC—a Dialect of ALGOL." *Communications of the ACM* 3, no. 8 (1960): 463–68.

Hutchins, William John. *Machine Translation: Past, Present, Future*. Chichester, UK: Ellis Horwood, 1986.

IBM Corporation. *Specifications for the IBM Mathematical FORmula TRANslating System FORTRAN*. New York: IBM, 1954.

_____. *Reference Manual FOR TRANSIT Automatic Coding System for the IBM 650 Data Processing System*. New York: IBM, 1957.

Jeon, Chihyung. "Flying Weather Men and Robot Observers: Instruments, Inscriptions, and Identities in US Upper-air Observation, 1920–1940." *History and Technology* 26, no. 2 (2010): 119–45.

Jesiek, Brent K. "Between Discipline and Profession: A History of Persistent Instability in the Field of Computer Engineering, circa 1951–2006" (Ph.D. diss., Virginia Polytechnic Institute and State University, 2006).

Johnson, Gerald J. "Of Metaphor and the Difficulty of Computer Discourse." *Communications of the ACM* 37, no. 12 (1994): 97–102.

Joint AIEE-IRE Computer Conference. *Review of Electronic Digital Computers: Joint AIEE-IRE Computer Conference—Papers and Discussions Presented at the Joint AIEE-IRE Computer Conference, Philadelphia, Pa., December 10–12, 1951*. New York: AIEE, 1952.

Kaiser, David. "Cold War Requisitions, Scientific Manpower, and the Production of American Physicists after World War II." *Historical Studies in the Physical and Biological Sciences* 33, no. 1 (2002): 131–59.

Kay, Lily. *Who Wrote the Book of Life? A History of the Genetic Code*. Palo Alto, CA: Stanford University Press, 2000.

Kline, Ronald. "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence." *IEEE Annals of the History of Computing* 33, no. 4 (2011): 5–16.

Knuth, Donald E. *Literate Programming*. Palo Alto, CA: Center for the Study of Language and Information, 1992.

_____, and Luis Trabb Pardo. "The Early Development of Programming Languages." In *A History of Computing in the Twentieth Century: A Collection of Essays*, edited by Nicholas Metropolis, Jack Howlett, and Gian-Carlo Rota, 197–273. New York: Academic Press, 1980.

Laning, J. Halcombe, and Neal Zierler. *A Program for Translation of Mathematical Equations for Whirlwind I*. Cambridge, MA: Instrumentation Laboratory/MIT, 1954.

Laszlo, Pierre. "Conventionalities in Formula Writing." In *Tools and Modes of Representation in the Laboratory Sciences*, edited by Ursula Klein, 47–60. Dordrecht, Netherlands: Kluwer Academic Publishers, 2001.

Lee, J. A. N. "Pioneer Day, 1982." *IEEE Annals of the History of Computing* 6, no. 1 (1984): 7–14.

_____, ed. "Special Issue, COBOL: 25th Anniversary." *IEEE Annals of the History of Computing* 7, no. 4 (1985).

Lehmer, Derrick H. "Welcoming Address." In *Proceedings of the Western Computer Conference, Los Angeles, California, February 11–12, 1954*, 7–8. New York: American Institute of Electrical Engineers, 1954.

Longo, Bernadette. "Metaphors, Robots, and the Transfer of Computers to Civilian Life." *Comparative Technology Transfer and Society* 5, no. 3 (2007): 253–73.

Luebbert, William F., and Percy W. Collom Jr. "Signal Corps Research and Development on Automatic Programming of Digital Computers." *Communications of the ACM* 2, no. 2 (1959): 22–27.

Mahoney, Michael S. "What Makes the History of Software Hard?" *IEEE Annals of the History of Computing* 30, no. 3 (2008): 8–18.

Martin, C. Dianne. "The Myth of the Awesome Thinking Machine." *Communications of the ACM* 36, no. 4 (1993): 120–33.

Martin-Nielsen, Janet. "Private Knowledge, Public Tensions: Theory Commitment in Postwar American Linguistics" (Ph.D. diss., University of Toronto, 2009).

"Mathematical Machine." *Science News-Letter* 48, no. 19 (1945): 291.

McLaren, Angus. *Reproduction by Design: Sex, Robots, Trees, and Test-Tube Babies in Interwar Britain*. Chicago: University of Chicago Press, 2012.

Mindell, David A. *Between Human and Machine: Feedback, Control, and Computing before Cybernetics*. Baltimore: Johns Hopkins University Press, 2002.

Mirowski, Philip, and Esther-Mirjam Sent. "The Commercialization of Science and the Response of STS." In *The Handbook of Science and Technology Studies,* 3rd ed., edited by Edward J. Hackett, Olga Amsterdamska, Michael E. Lynch, and Judy Wajcman, 635–89. Cambridge, MA: MIT Press, 2008.

Naur, Peter, J. W. Backus, F. L. Bauer, et al. "Report on the Algorithmic Language ALGOL 60." *Communications of the ACM* 3, no. 5 (1960): 299–314.

"News and Notices." *Communications of the ACM* 1, no. 2 (1958): 6–16.

Nofre, David. "Unraveling Algol: US, Europe, and the Creation of a Programming Language." *IEEE Annals of the History of Computing* 32, no. 2 (2010): 58–68.

Nye, David E. *Electrifying America: Social Meanings of a New Technology, 1880–1940*. Cambridge, MA: MIT Press, 1992.

Orgel, Sylvia. *Purdue Compiler General Description*. West Lafayette, IN: Purdue Research Foundation, 1958.

Paulsen, Gard. "Software in Telecommunications and the Programming Language Chili, 1974–1999" (Ph.D. diss., BI Norwegian Business School, 2011).

Perlis, Alan J. "Two Thousand Words and Two Thousand Ideas—the 650 at Carnegie." *IEEE Annals of the History of Computing* 8, no. 1 (1986): 42–46.

_____, and J. W. Smith. "A Mathematical Language Compiler." In *Automatic Coding: Proceedings of the Symposium Held January 24–25, 1957, at the Franklin Institute in Philadelphia*, edited by Franklin Institute, 87–102. Lancaster, PA: Franklin Institute, 1957.

_____, and Klaus Samelson. "Preliminary Report: International Algebraic Language." *Communications of the ACM* 1, no. 12 (1958): 8–22.

Pestre, Dominique. *Science, Argent et Politique: Un Essai d'Interprétation*. Paris: Quae, 2008.

Pflüger, Jörg. "Language in Computing." In *Experimenting with Tongues: Studies in Science and Language*, edited by Matthias Dörries, 125–62. Palo Alto, CA: Stanford University Press, 2002.

Priestley, Mark. *A Science of Operations: Machines, Logic and the Invention of Programming*. London: Springer-Verlag London Limited, 2011.

Pursell, Carroll W. "Technologies as Cultural Practice and Production." *Technology and Culture* 51, no. 3 (2010): 715–22.

Quarles, Donald A. "Need of Scientific Manpower." *Science News-Letter* 67, no. 10 (1955): 154–57.

Ransom, Harry H. "Scientific Manpower and National Security." *Journal of the American Society for Naval Engineers* 68, no. 4 (1956): 673–76.

Remington Rand Inc. "Preface." In *UNIVAC Short Code*. Philadelphia: Eckert-Mauchly Division/Remington Rand Inc., 24 October 1952.

_____. "Formation of USE—a Cooperative Organization of 1103A Users, 16 February 1956." *1103 Central Exchange Newsletter Number 8*, February 1956, 264–65, available at http://bitsavers.trailing-edge.com/pdf/univac/1103/PX71900-8_CentrExchNewsl%238_Feb56.pdf (accessed 15 December 2013).

Rogers, T. H. "Supply and Demand of Technical Personnel in American Industry." *School Science and Mathematics* 53, no. 2 (1953): 87–96.

Rosen, Saul, ed. "Programming Systems and Languages: A Historical Survey." In *AFIPS '64 (Spring): Proceedings of the April 21–23, 1964, Spring Joint Computer Conference*, 1–15. New York: ACM, 1964.

Rosenblueth, Arturo, Norbert Wiener, and Julian Bigelow. "Behavior, Purpose and Teleology." *Philosophy of Science* 10, no. 1 (1943): 18–24.

Rydell, Robert W. "The Fan Dance of Science: American World's Fairs in the Great Depression." *Isis* 76, no. 4 (1985): 525–42.

Sammet, Jean E. *Programming Languages: History and Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, 1969.

_____. "The Early History of COBOL." In *History of Programming Languages*, edited by Richard L. Wexelblat, 199–243. New York: Academic Press, 1981.

Scott, Allen J. "The Aerospace-Electronics Industrial Complex of Southern California: The Formative Years, 1940–1960." *Research Policy* 20, no. 5 (1991): 439–56.

Shannon, Claude E. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27, no. 3 (1948): 379–423.

_____. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27, no. 4 (1948): 623–56.

Shapin, Steven. "'The Mind Is Its Own Place': Science and Solitude in Seventeenth-Century England." *Science in Context* 4, no. 1 (1991): 191–218.

Simonson, G. R. "Missiles and Creative Destruction in the American Aircraft Industry, 1956–1961." *Business History Review* 38, no. 3 (1964): 302–14.

Smith, R. Blair. "The IBM 701—Marketing and Customer Relations." *IEEE Annals of the History of Computing* ("Special Issue on the IBM 701") 5, no. 2 (1983): 170–72.

Steel, T. B. "UNCOL: The Myth and the Fact." *Annual Review in Automatic Programming* 2 (1961): 325–44.

Stibitz, George R. "A Manual of Operation for the Automatic Sequence Controlled Calculator." *American Mathematical Monthly* 54, no. 1 (1947): 57–59.

_____. "The Organization of Large-Scale Computing Machinery." In *Proceedings of a Symposium on Large-Scale Digital Calculating Machinery*, 91–100. Cambridge, MA: Harvard University Press, 1948.

Strong, J., J. Wegstein, A. Tritter, J. Olsztyn, O. Mock, and T. Steel. "The Problem of Programming Communication with Changing Machines: A Proposed Solution." *Communications of the ACM* 1, no. 8 (1958): 12–18.

Tomalin, Marcus. *Linguistics and the Formal Sciences: The Origins of Generative Grammar*. New York: Cambridge University Press, 2006.

Turing, Alan M. "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society*, second series, 42 (1936–37): 230–65.

Tympas, Aristotle. "From Digital to Analogue and Back: The Ideology of Intelligent Machines in the History of the Electrical Analyzer, 1870s–1960s." *IEEE Annals of the History of Computing* 18, no. 4 (1996): 42–48.

U.S. Congress, Joint Committee on Atomic Energy. *Engineering and*

Scientific Manpower in the United States, Western Europe and Soviet Russia. Washington, DC: U.S. Government Printing Office, 1956.

U.S. Navy Mathematical Computing Advisory Panel, ed. *Symposium on Automatic Programming for Digital Computers, 13–14 May 1954*. Washington, DC: U.S. Department of Commerce/Office of Technical Services, 1954.

Weik, Martin H. *A Survey of Domestic Electronic Digital Computing Systems*. Ballistics Research Laboratories Report no. 971. Aberdeen, MD: Aberdeen Proving Ground, December 1955.

Wexelblat, Richard L. *History of Programming Languages I*. New York: Academic Press, 1981.

Wiener, Norbert. *Cybernetics or Control and Communication in the Animal and the Machine*. New York: John Wiley and Sons, 1948.

Wilkes, Maurice V., David J. Wheeler, and Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer, with Special Reference to the EDSAC and the Use of a Library of Subroutines*. Cambridge, MA: Addison-Wesley Press, 1951.

Williams, Samuel R. "Bell Telephone Laboratories' Relay Computing System." In *Proceedings of a Symposium on Large-Scale Digital Calculating Machinery*, 41–68. Cambridge, MA: Harvard University Press, 1948.

Wolfe, Ruth W. "The Technical Manpower Shortage." *School Science and Mathematics* 57, no. 1 (1957): 63–70.