Reconsidering the Stored-Program Concept

Thomas Haigh
University of Wisconsin–Milwaukee

Mark Priestley
Crispin Rope

The first in a three-part series appearing in *IEEE Annals*, this article gives a historical explanation of the endemic confusion surrounding the stored-program concept. After a detailed investigation of the history of this idea, the authors propose three more precisely defined alternatives to capture specific aspects of the new approach to computing introduced in 1945 by John von Neumann and his collaborators.

It is a truth universally agreed that implementation of the "stored program concept" in the late 1940s was the most important dividing line in computer history, separating modern computers from their less evolved predecessors. Yet, as Doron Swade recently noted, we do not really agree on why this should be the case. For years he "assumed that the significance of the stored program must be self-evident" and attributed his own confusion to "a deficiency of understanding" or to "some lack" in his computer science education, until finally he "became bold and began asking" among computer historians and pioneers what it actually was. Their answers were "all different," with the question of whether "the primary benefit was one of principle or practice frustratingly blurred." Swade concluded that,

> There was one feature of all the responses about which there was complete agreement: no one challenged the status of the stored program as the defining feature of the modern digital electronic computer. . . . While the reasons given for this were different, none discounted its seminal significance. But it seems that we struggle when required to articulate its significance in simple terms and the apparent mix of principle and practice frustrates clarity.[1]

In this article, we respond by historicizing the "stored program" and "stored program concept." Historians almost invariably point to a single document as the first publication to describe the concept and as the direct source of inspiration for the architecture of subsequent computer projects. That document is the "First Draft of a Report on the EDVAC" (hereafter simply the First Draft), circulated under the name of John von Neumann in 1945.[2] Although the true balance of credit for the ideas contained in this document has been widely and heatedly debated, its central importance to the development modern computing has not.[3]

We look at initial conceptions of the advantages and crucial features of the new approach to computer design put forward in the First Draft, which scholars treat as the first and most influential statement of the concept. We also identify the origins of the phrase "stored program," which came some years later, and its extension by early computer historians to the "stored program concept."

Having shown that this historical evolution left the terms hopelessly overloaded with contradictory meanings, to which Swade's confusion was an appropriate and insightful response, we return to the text of the First Draft to identify three distinct but intertwined clusters of influential ideas in the report: the modern code paradigm, the von Neumann architecture paradigm, and the EDVAC hardware paradigm. These were sometimes implemented independently in

 Published by the IEEE Computer Society 1058-6180/14/$31.00

the machines of the 1940s, and although all three were standard features of computers of the mid-1950s, their fates have subsequently diverged again. We believe that reliance by historians on the term "stored program concept" as shorthand for the content of the entire 1945 EDVAC design has done more to hurt our understanding than to help it.

Our own attention turned to this question as we investigated modifications made to ENIAC in early 1948 by a team working closely with John von Neumann. These modifications incorporated key elements of the new approach to computer design and programming associated with EDVAC. Thus converted, ENIAC ran a complex program written in the new style, including conditional branches, data reads from calculated addresses, and a subroutine called from more than one point in the code. This program was developed using the methodology and flow diagramming notation described by Herman Goldstine and von Neumann in their seminal series of reports on "Planning and Coding of Problems for an Electronic Computing Instrument" issued around this time.[4]

Dispelling some previous confusion, we have established that all of this was completed before Manchester University's "Baby" (known more formally as the Small-Scale Experimental Machine) executed what is usually called the world's first stored program in the summer of the same year. It would be well over a year until EDSAC became the first purpose-built stored-program computer to enter regular operation and run programs of complexity comparable to ENIAC's. ENIAC still differed from purpose-built, full-scale stored program computers in several important ways. In particular, its electronic storage remained rather small, and program instructions were stored along with constant data in a high-speed read-only memory.[5] Should ENIAC therefore be considered the first operational stored-program computer? Well, it all depends on what we mean by "stored program." So before returning to ENIAC in the next article in this series, we will clarify the ideas associated with this term and the process by which they evolved.

To understand the role played by the First Draft in the subsequent development of computing, we are relying on the notion of a "paradigm" introduced by historian and philosopher of science Thomas Kuhn in his classic *Structure of Scientific Revolutions*.[6] Kuhn believed that the most fundamental sense of paradigm, the seed around which mighty scientific communities grew, was an exemplary technical accomplishment reflecting a new approach—the "concrete" paradigm. In its initial formulation it might be clumsy or incomplete, but it held sufficient promise to attract others to build on its model to extend it and apply it to new kinds of problem. The original paradigmatic accomplishment was, in Kuhn's term, "articulated" through this later work to become something almost unrecognizable. For example, later generations of scientists learned Newton's laws of motion in a form (and using a version of calculus) quite different from those familiar to Newton himself.

Focusing on the First Draft as a paradigm thus steers us toward two important issues. The first is that this core sense of paradigm helps us to understand the enormous power that the First Draft exerted over the subsequent development of computing. The second, less obvious, is that the First Draft only became a paradigm retroactively. In 1945 it was just a document. As others took up its ideas, implemented them, and extended them, it came to function as a paradigm. By the 1950s, its paradigmatic authority was becoming clear. Some of the ideas contained within it were discarded, some were reformulated, and others were added. The treatment of its ideas in textbooks and papers has continued to evolve. Understanding what was so originally so important about the First Draft requires us to strip away some of these later ideas and to ground our analysis in the realities of 1940s computer practice.

What Did von Neumann Mean By "Stored Program"?

That turns out to be a trick question, unfortunately. The "First Draft of a Report on the EDVAC" does not, despite the role it has been assigned in later historical work, function well as a standards document to rigorously define the concept of "stored program." To begin with, the word "program" never appears in the draft. Von Neumann consistently preferred "code" to "program" and wrote of "memory" rather than "storage." Indeed, if someone with no prior exposure to the topic was handed this document and asked to encapsulate its big idea about the handling of instructions in a single phrase, he or she would be much more likely to derive "remembered code" than "stored program."

Our current attachment to the term "stored program" as a description for computers built along lines proposed for EDVAC

thus needs some historical explanation. Read literally, the term conveys very little. Any program that can be executed by a computer must be stored in some form or other. The First Draft itself observed that "instructions must be given in some form which the device can sense: Punched into a system of punch-cards or on teletype tape, magnetically impressed on steel tape or wire, photographically impressed on motion picture film, wired into one or more fixed or exchangeable plugboards—this list being by no means necessarily complete."[7]

The report was a speculative and argumentative description of the design of a particular machine, not an abstract description of a new class of machines. It offered great detail in some areas and very little in others. In other words, it is only when armed with the knowledge of later developments that we might hope to identify a specific paradigm in there.

It is indeed true that the First Draft argued for the collocation of code and data, often taken to be the essence of the stored-program concept, although von Neumann's language introducing the idea was uncharacteristically tentative: "While it appeared that various parts of this memory have to perform functions which differ somewhat in their purpose, it is nevertheless tempting to treat the entire memory as one organ, and to have its parts as interchangeable as possible."[8] Von Neumann believed that the data requirements of the problems he was interested in were large enough to require a memory of unprecedented size and that the program code would, by comparison, be quite small. Using one set of mechanisms to handle both would simplify EDVAC.

Two intertwined histories are of interest here. The first is the process by which the many different ideas explored in the First Draft and related documents were reduced to a shared understanding that the storage of code and data in a shared memory was the single crucial shift setting the late-1940s generation of computers aside from their predecessors. This has received little attention from historians. The second is the process by which the specific term "stored program" was attached to this concept. That has received no attention whatsoever.

Early Understanding of the EDVAC Design

Some of the earliest and most perceptive comments on the essential features of the new breed of computers came from J. Presper Eckert and John Mauchly. As well as designing ENIAC, they had created at least some of the ideas included in the First Draft and in 1946 founded a company to commercialize a similar design. In 1946 a summer school was held at the University of Pennsylvania, then housing both ENIAC and the project to build the EDVAC. Historians have credited the event as a crucial vector for the spread of the stored-program concept. But what made EDVAC so appealing as a model?

During his lecture, Eckert retraced for the audience the process he and his colleagues had gone through to design the EDVAC as a reaction to the shortcomings of ENIAC. His argument for internal program storage was pragmatic. It would reduce set up time, as "in the EDVAC there will be no cords, no plugs, and few switches. We are simply going to use the memory to hold the information electronically and to feed those pieces of information which relate to programming from the memory, into the control circuits in order to resequence the machine." ENIAC had used a vast number of vacuum tubes to create a writable electronic memory of just 200 decimal digits. This technology could never be used to create a memory large enough to store all the data needed for many problems, let alone programs. He estimated that the connecting cables, function tables (large banks of switches originally intended to hold tables of numerical values to lookup function values from), and switches of ENIAC effectively held another 7,000 or so digits of program and data information. But with a larger, cheaper memory of the kind offered by his recent invention of the mercury delay line, there was no longer any reason to store this program information separately. Eckert noted that the ratio between program and data storage varied from problem to problem, so that combining the two "eliminates for the designer the problem of attempting to find the proper balance between the various types of the memory, and gives the problem to the user."[9]

Memory technologies were, as these comments remind us, the dominant challenge facing computer builders in the late 1940s. Discussions of drums, delay lines, Selectrons, cathode ray tubes, wire recorders, and phosphor discs occupied a central place in the first computing texts and conferences. The success or failure of the various teams rushing to make their computers operational was determined in large part by their success in

disciplining unruly cathode ray tube and delay lines. Avoiding design complexity meant cheaper, faster construction and improved reliability.

John W. Mauchly, Eckert's collaborator, offered similar sentiments at the Harvard University symposium held the next year. The title of his paper, "Preparation of Problems for EDVAC-Type Machines," indicates the kinds of terminology used before "stored program" gained currency. Mauchly considered the "fundamental characteristics … which differ significantly from present machine design…. [T]hree have a definite bearing on the handling of problems: (1) an extensive internal memory; (2) elementary instructions, few in number, to which the machine will respond; and (3) ability to store instructions as well as numerical quantities in the internal memory and modify instructions so stored in accordance with other instructions." Expanding on the final point, he noted that,

> [I]nstructions are stored in the internal memory in the same manner as are numerical quantities, and one set of instructions can be used to modify another set of instructions. This is directly related to the finite capacity of the internal memory and the limited number of basic operations…. The total number of operations for which instructions must be provided will usually be exceedingly large, so that the instruction sequence would be far in excess of the internal memory capacity. However, such an instruction sequence is never a random sequence, and can usually be synthesized from subsequences which frequently recur.
>
> By providing the necessary subsequences, which may be utilized as often as desired, together with a master sequence directing the use of these subsequences, compact and easily set up instructions for very complex problems can be achieved. Even greater powers are conferred, however, by the ability to use one instruction to modify another…. [This] transfers to the machine a burden which would otherwise fall upon the operator—the burden of explicitly writing out and coding the successive variations which are to be used.[10]

In his 1949 book *Calculating Instruments and Machines*, the eminent British mathematician Douglas Hartree treated ENIAC, Harvard Mark I, and SSEC as "The First Stage of Development" of "Large Automatic Digital Machines." Hartree concluded that,

> [I]t seems very improbable that any of them will be duplicated. The machines of the future will be considerably different in principle and appearance; smaller and simpler, with … tubes and relays numbered in thousands rather than tens of thousands of the machines considered in this chapter, faster, more versatile and easier to code for and to operate. Those at present projected or under construction are different enough to be regarded as forming a second stage of development.[11]

Taken together, the observations of Eckert, Mauchley, and Hartree help us to understand the 1945 EDVAC design as a reaction among the Moore School team against the complexity that had plagued ENIAC. The vacuum tubes composing the logic units of early electronic computers were bulky and unreliable and posed a significant risk of fire. ENIAC included almost 18,000 such tubes. EDVAC-type computers of significantly greater power and flexibility reduced this number by a factor of about five, just as Hartree had promised. EDSAC had around 3,000 tubes, as did the Institute for Advanced Studies computer. The Manchester Mark 1 had around 1,300 and the Pilot ACE just 800.[12]

The lion's share of this reduction came from avoiding ENIAC's use of 11,000 vacuum tubes for its cramped high-speed memory. However, the elimination of hardware did not stop there. Each of ENIAC's accumulators included circuitry to perform additions. These duplicate circuits were replaced by a single central adder. As noted in the First Draft, "The device should be as simple as possible, that is, contain as few elements as possible. This can be achieved by never performing two operations simultaneously."[13] Von Neumann admitted that "up to now all thinking about high speed digital computing devices has tended in the opposite direction," but he felt this justified applying his new "uncompromising solution … as completely as possible" until experience might prove that compromise was necessary.

In conclusion, computing experts of the late 1940s spoke of "EDVAC-type" computers rather than stored-program computers, reflecting the role of the First Draft report as a paradigmatic exemplar. Contemporary discussion focused on the novelty of electronic machines with large high-speed memories,

new programming methods, and relatively simple logic units. Experts did come, quite quickly, to identify the storage of programs and data in a large electronic memory as one of several central innovations of EDVAC-like machines. Some also noted the benefits of allowing programs to modify themselves during execution. However the interchangeability of programs and data was a single contribution to a much larger and more radical simplicity of design in comparison with machines such as ENIAC and SSEC.

Origins of the Term "Stored Program"

The proceedings of the Moore School lectures, issued in 1948, did not include the phrase "stored program."[14] Neither could we find it in the proceedings of a conference held at Harvard in 1947, the introductory books published by Hartree in 1947 and 1949, the proceedings of the Cambridge University computer conference held in 1949, or the ERA guide published in 1950.[15] In fact, we have been unable to locate the phrase in any publication of the 1940s.

During the second half of the 1940s, authors used a variety of terms to describe the new breed of computers. The most common was "digital automatic computer." "Digital" separated them from analog machines such as differential analyzers. "Automatic" made it clear that machines, rather than people, were being referred to. "Electronic," another popular adjective, set the new high-speed machines apart from their electromechanical ancestors. In these terms, the distinction between computers such as ENIAC or IBM's SSEC and computers patterned after EDVAC was not always apparent.

So where did "stored program" actually come from, and why did it eventually replace alternatives such as "EDVAC-type machine" as a description of the new kind of computer? The earliest use we have been able to establish is in 1949, within the small team at IBM's Poughkeepsie facility, which under the direction of Nathaniel Rochester, produced IBM's first EDVAC-type computer, usually called the Test Assembly. This experimental system was built around the firm's first electronic calculator, its 604 Electronic Calculating Punch, which became the arithmetic unit of a lashed-up computer. It was joined to a new control unit, a cathode ray tube memory, and a magnetic drum.

The resulting machine had two potential programming mechanisms, as the 604 already included a plug board able to hold a program of up to 60 instructions. To distinguish between this wired program and the more complex and flexible sequence of instructions held in the 250-word electronic memory or on the drum, the team began to call the latter the "stored program." A proposal written by Rochester in 1949 noted that the cost of the plug board and the work required to program it became impractical once a certain level of complexity had been reached. Thus, "the best solution to this difficulty is to introduce the calculating program into the machine on a deck of tabulating cards and to retain it, along with the numerical data, in the storage section of the calculator."[16] The report was titled "A Calculator Using Electrostatic Storage and a Stored Program."

Rochester and his collaborators took the term "stored program" with them as they moved from designing the Test Assembly to its experimental successor, the Tape Processing Machine, and eventually to the IBM 701 (the firm's first computer product). In later usage, this discussion of digital computers with "stored program control" collapsed simply into discussion of "stored-program computers." An early example comes in a 1951 paper presented at the Joint Computer Conference by two IBM employees describing its Card Programmed Calculator, a product coupling the 604 with a card-driven control unit. The authors praised the flexibility and speed of this approach versus a "stored program machine" for which "it is usually necessary to economize on the length of sequences [of instructions], on account of the limited storage available."[17]

In 1953 the term was sufficiently well known within the small world of electronic computer users that Willis Ware of the Rand Corporation could write simply of "what we know now as the 'stored program machine'" in a report on the "History and Development" of von Neumann's computer project.[18] It was not enormously common, but it did occur with reasonable frequency in the conference proceedings of the 1950s, particularly in presentations by IBM staff. However, "stored program" did not seem to be a part of the firm's carefully controlled official vocabulary: we have not come across it in documentation or advertisements for IBM's early computers.

"Stored Program" Becomes a Historical Term

After establishing this limited beach head in the 1950s, the phrase "stored program" does

not appear to have made significant advances in the 1960s, probably because all digital computers of the era executed their programs from memory rather than reading them one instruction at a time from an external source. In the title of a book or article, the term "digital computer" was understood to imply stored-program control.

Only with the rise of interest in the history of computing did it again become necessary to distinguish stored-program computers from other kinds of digital computers. Discussions of "stored program" enjoyed something of a renaissance in the 1970s, beginning with its prominent use by computer pioneer Herman Goldstine in his book, *The Computer from Pascal to von Neumann*.[19] This remained the most important overview of computing history for many years, used as a textbook into the 1990s. Goldstine, then an IBM fellow, helped to establish this relatively obscure technical term at the center of the growing historical discourse.

Eminent computer scientists and their professional organizations conducted most of the initial work in this field. Their main focus was the origin of computer technology, and a great deal of effort was devoted to documenting facts about early computers, both famous and obscure. One question loomed above all others: what was the first computer? Among scholarly historians, in contrast, the question of the "first computer" is no longer seen as legitimate. The answer to the question depends almost entirely on how one defines "computer." Introducing a conference devoted to early computers, Michael R. Williams suggested that his colleagues "not use the word 'first'—there is more than enough glory in the creation of the modern computer to satisfy all of the early pioneers, most of whom are no longer in a position to care anyway."[20] In the same address, Williams noted that, "If you add enough adjectives to a description you can always claim your own favorite. For example ENIAC is often claimed to be the 'first electronic, general purpose, large scale, digital computer' and you certainly have to add all those adjectives before you have a correct statement."[20]

This consensus on the appropriate title for each machine is relatively recent, reflecting a kind of truce reached during the 1980s. The separation of "stored program computer" from "general purpose computer" was initially contested. Arthur and Alice Burks, in a substantial 1981 article documenting ENIAC, had attempted to precisely define the capabilities of a general-purpose computer. In response Brian Randall, an early leader of historical efforts in the field, suggested that ENIAC could not be considered general purpose because it lacked the crucial feature of being "able to select among items held in its read-write memory, based on results so far computed," which was "one of the most significant and distinct characteristics of so-called stored-program computers."[21] In other words, to Randell only an EDVAC-type computer could be considered general purpose.

Assigning the Firsts

After historians collectively withdrew the prize of "first computer" without making an award, the most dazzling of the remaining trophies bore the legend "first stored program computer." This put machines like ENIAC, programmed with wires and switches, or the Harvard Mark I calculator, which read its instructions one a time from a paper tape, on the wrong side of the historical divide. The disagreements noted by Swade were not a barrier to acceptance of this distinction, and may even have been an advantage. The practical purpose of these distinctions was to establish a truce between advocates for a handful of different computers. With so few machines to distinguish between, the community could agree that, for example, the Mark I was not a stored-program computer and EDSAC was, without having to reach consensus on exactly what permutation of the many differences between them was necessary and sufficient to accord this status to the later machine. Of course, the new consensus did imply that the biggest dividing line between the modern computer and its antecedents was in its ability to load a data sequence into memory and execute it as a program, but the extent to which this was seen as the only significant difference or as one aspect of a much broader shift could and did vary wildly. This explains the disagreements Swade encountered as to the significance of the stored program.

The figurative trophy for "first stored program computer" was metaphorically cut in half and split between two British computers: Cambridge University's EDSAC and Manchester University's Baby. The Baby, which ran its first program on 21 June 1948, was only a small-scale prototype. Its primary function was to test the workability of a novel method of using a cathode ray tube for data storage. It was put together quickly with the most modest configuration possible—just

eight instructions (not including add) and a single memory tube holding 32 words of memory. It ran several test programs to prove the reliability of the memory but never tackled a program of any practical use before it was disassembled and its parts were redeployed build a full-sized computer.

In contrast EDSAC was, by the standards of its day, a powerful computer. After it ran its first programs on 6 May 1949, it remained in service until 1958. Its creators built the first subroutine library, wrote the first programming textbook, and pioneered several key ideas in systems programming including the assembler. EDSAC was, like ENIAC, applied to numerous scientific and mathematical problems. In their authoritative overview of the history of computing, Martin Campbell-Kelly and William Aspray wrote that, with the first successful run of EDSAC in May 1949, "[t]he world's first practical stored-program computer had come to life, and with it the dawn of the computer age."[22]

The Stored-Program Concept Meets Computer Science

The resurgence of the stored-program concept, now as a concept for historical discussion, went along with its increasing identification with foundational ideas from the new discipline of computer science. In recent decades, the manipulation of programs and data interchangeably in the same memory units has increasingly been taken as the key defining characteristic of the stored-program computer, and thus of modern computers. This was a third stage in the evolution of the term "stored program." We saw that this term originally described a type of program but was soon used to define a class of "stored program computers." Historical discussion extended this further with the implication that a "stored program concept" was the dividing line between these machines, the first true computers, and their predecessors. Thus the complex legacy of the EDVAC design described in the First Draft was rhetorically condensed to a single idea. In turn, the stored-program concept and general-purpose computer have sometimes conflated with the more formal concept of a computer being Turing complete or "universal" if equipped with a memory of infinite size. This has led some to claim that Turing was the true inventor of the stored-program computer![23]

To cite just three of many recent examples of this conventional wisdom: the Wikipedia page on "stored program computer" currently defines it as "one which stores program instructions in electronic memory. Often the definition is extended with the requirement that the treatment of programs and data in memory be interchangeable… [t]he stored program computer idea can be traced back to the 1936 theoretical concept of a universal Turing machine."[24] In his recent *Computing: A Concise History,* Paul Ceruzzi defined stored-program computers as storing "both their instructions—the programs—and the data on which those instructions operate in the same physical memory device" and suggested that this "extended Turing's ideas into the design of practical machinery."[25] Finally, Swade himself retreated from the endearingly bold confession of confusion we quoted earlier to conclude that "the internal stored program … is the practical realization of Turing universality" and thus conferred "plasticity of function, which in large part accounts for the remarkable proliferation of computers and computer-like artifacts."[26]

Arguing about the influence Turing might or might not have exerted over von Neumann has become an enjoyable parlor game for historians of computing. That question aside, one will find few references to Turing's theoretical work among the discussions of those building computers in the 1940s.[27] Some historians have suggested that the retroactive embrace of Turing as a foundation for this practical work is tied to the emphasis within computer science, as it emerged as a distinct discipline during the late 1950s and 1960s, on abstract models of computation.[28] In later discussions, the advantages of stored-program machines were often justified according to the theoretical concerns of academic computer scientists rather than the pragmatic issues of primary importance to their designers.

Self-Modifying Code

One of the major theoretical attractions of storing code and data in the same writable memory is that a program can manipulate its own code as it is being executed, using the same mechanisms with which it manipulates other kinds of data. On one level, this emphasis on code modification is startling. Writing programs that modify their own code at runtime has been seen for decades as a serious breach of programming etiquette. The capability is sacrificed in most high-level languages and thus has become increasingly remote from mainstream programming practice.

Instruction modification would have been a common operation in programs written for the EDVAC described in the First Draft. It was the only way to terminate a loop, perform a conditional branch, or alter the address from which an instruction fetched data (for example to obtain a value from a different cell within an array each time a block of code is looped through).[29] What these applications have in common is that it is not necessary to alter the operation code itself (for example, to change a subtract instruction to a jump instruction) but merely to change the field within the instruction that specifies the memory address from which data should be fetched or to which a jump should be made.

Von Neumann deliberately eliminated the possibility of a program overwriting itself. One of the 32 bits in each word of memory flagged it as holding either program or data. A transfer operation applied to a data word would fully overwrite its content. Applied to an instruction word, the same operation would overwrite only the address field.[30] The stipulation was dropped from later designs for EDVAC-like machines, including the design produced by von Neumann's own group at the Institute for Advanced Studies.

We see the reliance placed on address modification in the 1945 EDVAC design as an expression of a broader determination to radically simplify ENIAC's successor by replacing ENIAC's many special-purpose mechanisms with a small number of general-purpose mechanisms. ENIAC in its original configuration relied on special hardware to coordinate loops, including a dedicated "Master Programmer" unit holding 10 electronic "steppers" and a set of electronic counters constituting a special purpose memory inaccessible to the rest of the machine. Another special-purpose control mechanism allowed switches on other units to repeat operations up to nine times. Conditional branches were accomplished with a complex technique using special adaptors to route numerical data into program control lines. In the new design, branching and looping were both accomplished with a simple control transfer instruction. Like many early computers, ENIAC had included dedicated hardware for table lookups. Its function tables were designed to return the appropriate value of a function after being sent an "argument" stored in an accumulator. The argument would change as the computation continued, and to simplify the calculation of intermediate values of the tabulated function using interpolation, another special feature allowed easy lookup of neighboring values. In the new paradigm, the same device was reinterpreted as a general-purpose read only memory and the "argument" became an "address."

As described in the First Draft, the EDVAC would have no conditional branch instruction, no special support for loops, and no indirect addressing capabilities to read or write from a different part of a table every time a loop was executed. All would be accomplished by explicitly modifying the address portions of instructions held in memory. In this regard, the First Draft design for the EDVAC went too far in its determination to replace the gothic excess of ENIAC with an austere minimalism. Early stored-program computers retained much of its radical simplicity. However, they invariably included special instructions for conditional branches (used to terminate loops, among other things) and the Manchester Mark 1's addition of an index register for relative addressing was widely copied. These features added slightly to hardware complexity but greatly reduced the need for self-modifying code and so made programs easier to write, easier to debug, smaller, and faster.

Instruction modification is the most dramatic example of the way in which later discussions of the "stored program concept" required a selective reading of the 1945 First Draft, one heavily influenced by knowledge of later developments. Despite the paradigmatic position of the First Draft, we pick and choose the features to take seriously from it. Still, if one accords the First Draft its conventional status as the seminal statement of the stored-program concept, then unrestricted code modification is clearly not part of said concept.[31]

Code modification ultimately found its niche as a technique for systems software rather than for looping, branching, or addressing. As Alan Bromley once pointed out, a loader or bootstrap program has to be able to insert instructions into memory, and any kind of operating system relies on the ability to overwrite program code currently in memory.[32] The main practical use of the capability has therefore been in the development of systems software, a concept that did not exist in 1945.

Defining the "Modern Code Paradigm"

Within the past few years a new generation of scholarly historians has turned back to the

events of the 1940s with different perspectives and questions. Allan Olley has explored the claims of the IBM SSEC to be the first operational stored-program computer.[33] Liesbeth de Mol and Marteen Bullynck have delved deep into the programming practices adopted by mathematicians confronted with the complexity of ENIAC in its original programming mode.[34] One of us, Mark Priestley, has broken the stored-program concept down into a number of distinct innovations, realized at different points in the 1940s.[35] On balance, however, we feel that the historical baggage accumulated by the "stored program concept" means that historians should treat it with much the same wariness we have learned to associate with "first computer." The time has come to replace it, as an analytical category, with a set of more specific alternatives amenable to clear and precise definition.

The first of these is the "modern code paradigm." We use this new term to describe the program-related elements of the 1945 First Draft design that become standard features of 1950s computer design. Some items specified in the report were ignored or changed by actual computer designers, while some common code capabilities of 1950s computers came from other sources. The first computers modeled on the EDVAC design differed in many ways from the design sketched in the 1945 report and from each other. These differences included memory size and characteristics, instruction format, addressing modes, and treatment of code modification. EDVAC, as eventually constructed, used a four-address instruction format. The Pilot ACE did not use operation codes at all, instead expressing each instruction using a system of logical "sources" and "destinations." Real computers relied much less heavily on code modification than the 1945 design for EDVAC had done; they supported conditional jumps via special instructions rather than code modification, and within a few years, new addressing modes greatly reduced the need for address modification during data access. The Manchester computers pioneered this; the Baby performed all addressing indirectly, while the Mark 1 is famous for having introduced the index register.

Looking for novel code-related features from the 1945 First Draft that had become taken-for-granted features of computers a decade later therefore illuminates the process by which a sprawling, idiosyncratic, and brilliant document became a dominant paradigm for the builders of computers. Not all of the following features were original to the First Draft, but their packaging together and integration with the von Neumann architecture and new hardware technologies (discussed later) exerted a remarkable influence.[36]

1. *The program is executed completely automatically.* To quote the First Draft, "[o]nce these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention." This was essential for electronic machines, whereas human intervention at branch points had been workable with slower devices such as the Harvard Mark I. Of course, operators still had to tend to input and output devices, and data might require preprocessing and postprocessing, either manually or with punched card equipment.
2. *The program is written as a single sequence of instructions.* This sequence of instructions is referred to as EDVAC's "orders" in the First Draft, and stored in numbered memory locations along with data. These instructions control all aspects of the machine's operations. The same mechanisms are used to read code and data. As discussed earlier, the First Draft specified the explicit demarcation of memory locations holding code from those holding data.
3. *Each instruction within the program specifies one of a set of atomic operations made available to the programmer.* This was usually done by beginning the instruction with one of a small number of operation codes. Some operation codes are followed by argument fields specifying a memory location with which to work or other parameters. The First Draft specified just seven "types of orders" to be coded with three bits, although four of these instructions also included an additional four bits as a kind of parameter to select one of 10 arithmetic and logic operation.[37] Several order types additionally incorporated 13-bit addresses. All together, orders required between 9 and 22 bits to express. Actual machines usually followed this pattern, typically merging the "order type" and "operation" fields from the First Draft so that each arithmetic or logical operation received its own numerical order code.

The main exception comes with Alan Turing's ACE design and its derivatives, which stuck close to the underlying hardware by coding all instructions as data transfers between sources and destinations.

4. *The program's instructions are usually executed in a predetermined sequence*. According to the First Draft, the machine "should be instructed, after each order, where to find the next order that it is to carry out." In the EDVAC this was to be represented implicitly by the sequence in which they were stored, as in "normal routine" it "should obey the orders in the temporal sequence in which they naturally appear." It also pointed toward the idea of a program as a readable text: "it is usually convenient that the minor cycles expressing the successive steps in a sequence of logical instructions should follow each other automatically."
5. *A program can instruct the computer to depart from this ordinary sequence and jump to a different point in the program*. As the First Draft puts it, "[t]here must, however, be orders available which may be used at the exceptional occasions referred to, to instruct CC to transfer its connection to [i.e. fetch the next instruction from] any other desired point in M [memory]." This provided capabilities such as jumps and subroutine returns.
6. *The address on which an instruction acts can change during the course of the program's execution*. That applies to the source or destination of data for calculations or the destination of a jump. This address modification capability was expressed rather cryptically in the First Draft, the final sentence of which noted that when a number was transferred to a memory location holding an instruction only the final 13 digits, representing the address $\mu\rho$, should be overwritten. Actual computers achieved this instead through unrestricted code modification and/or indirect addressing mechanisms. EDVAC would have relied on address modification to make a conditional jump (for example, to terminate a loop), but the designers of actual machines recognized the importance of this operation and gave it a special instruction.[38]

A consequence of these features was that the logical complexity of the program was limited only by memory space available to hold instructions and working data. This contrasted with the dependence of machines such as the original ENIAC or SSEC on other resources such as program lines, plug board capacity, or number of tape readers as determinants of logical program complexity.

As we explained earlier, we do not view the modern code paradigm as a new name for the "stored program concept" or as an idea encompassing the full scope of meanings associated with the latter. Indeed, the more specific scope of the new term is a large part of its appeal. There were clearly several other aspects of the First Draft and subsequent publications by members of von Neumann's group in Princeton that had a major influence on later computer builders.

To adapt an existing term, the second facet we identify in the First Draft might be called the "von Neumann architectural paradigm." This includes the basic structure of "organs" found in the report, including the separation of memory from control and arithmetic. Associated with this are the serialization of computation, with only one operation taking place at a time; the routing of all memory transfers through the central arithmetic unit; and the system of special-purpose registers that serve as source and destination for arithmetic and logic instructions and provide a program counter and instruction register for control purposes. The von Neumann architecture has in general been more clearly defined within the technical literature than has the stored-program concept. One might, of course, dispute the extent to which it is fair to attach only von Neumann's name to these concepts. That has little bearing on our argument here, but we note that "EDVAC architecture paradigm" could serve as an alternative.

The third major facet might be termed the "EDVAC hardware paradigm." The EDVAC approach appealed to early computer builders in large part as a way of building powerful, flexible machines using a relatively small number of components. Influential hardware ideas in the First Draft include the use of delay line or storage tube memory, building logic entirely from electronic components, representing all quantities in binary, and keeping special-purpose or duplicate hardware mechanisms to a minimum. (Von Neumann considered that a multiplier would justify itself but that duplicating adders or providing hardware for more specialized functions would provide little benefit.) These hardware features were not entirely

unknown, with the possible exception of the memory technologies, but collectively they represented a bold commitment to new technologies at a time when computing groups within Harvard, Bell Labs, and IBM were still drawing up plans for new high-end machines based on relay storage and paper-tape control. Thus, we believe that the hardware choices specified for EDVAC in the First Draft function as a paradigm, in Kuhn's core sense of a powerful and tangible exemplar.

These three paradigms have intertwined early histories but were always at least partially separable and ultimately diverged. The machines of the mid-1950s tended to implement all three. In the 1940s, we can see some interesting divergences, as machine builders would pick and choose among them. Alan Booth's ARC followed both the modern code paradigm and the von Neumann architecture but implemented them using relay hardware. Martin Campbell-Kelly observed that Booth's claimed operation date of 12 May 1948 would make this "the first operational EDVAC-type stored program computer (although it was not of course electronic)."[39] Alan Turing's design for the ACE adopted much of von Neumann's architecture and followed EDVAC's hardware paradigm but relied on a different kind of instruction format with no conventional operation codes. As Campbell-Kelly noted, "Most computers are sufficiently alike that a knowledgeable programmer can get a fairly good appreciation of a machine from its instruction format and a table of operation codes. The Pilot ACE is an exception."[40]

Most saliently for our broader project, ENIAC after its conversion followed the modern code paradigm with surprising faithfulness. The feel and structure of its program code bears an unmistakable kinship with those produced for other early machines sharing the same model. Its use of indirect addressing to accomplish conditional jumps and other operations for which EDVAC would have relied on the direct modification of stored instructions is a variation of code style within the paradigm rather than a fundamental divergence. ENIAC was all-electronic but clearly did not reflect other aspects of the EDVAC hardware paradigm. In particular, it still had a much higher number of vacuum tubes than its immediate successors, while suffering from an unusually small writable memory.

Our work involves a shift of analytical priorities from theory to practice and from design to use. Early computing primers explained EDVAC-type machines to potential computer users primarily by documenting a sample instruction set rather than articulating architectural checklists. From this viewpoint, the storage of a program in read-only or rewritable memory is significant to the extent that it dictates a major change in programming style or imposes arbitrary limits on computational capabilities or program complexity.

The paradigmatic influence of these three aspects of EDVAC diverged again after the mid-1950s. The relevance of von Neumann's EDVAC design as a hardware paradigm faded first, as transistors and core memories made vacuum tubes and delay lines obsolete. The von Neumann architectural paradigm enjoyed a longer life, although innovations such as parallel processing, message passing interfaces, instruction pipelining, direct memory access by peripherals, stacks, and addressable registers gradually erased its radical minimalism. In contrast, the modern code paradigm has remained largely intact, at least as a description of the machine language executed by processors. It was extended and made more specific in many ways, not the least by von Neumann's own 1946 description of the planned structure of the Institute for Advanced Studies machine.[41] It was not, however, overturned.

By the mid-1950s, it was already becoming unusual for this code to be written directly by humans. The computer itself was increasingly relied on to automate aspects of program preparation. This was not anticipated in the First Draft, which said little about programming. To some extent, the "Planning and Coding" reports produced by von Neumann and Goldstine in 1947 and 1948 served as a model for these developments, as did the programming textbook issued by Wilkes, Wheeler, and Gill.[42] However, the historical record does not make it easy to identify any single dominant approach to programming itself or to point to any single document as a paradigmatic statement of programming practice.

Conclusion

Answering Swade's challenge has convinced us that the dominant understanding of what the stored-program concept is, and of why it is important, has changed considerably over time. Having given a historical explanation of the endemic confusion surrounding the stored-program concept, we suggested that

historians returning to the era with an eye to the investigation of programming practice and computer use are likely to find the concept unhelpfully imprecise and overloaded with contradictory associations. Instead, we suggest the adoption of more precisely defined alternatives to capture specific aspects of the new approach to computing associated with the work of von Neumann and his collaborators.

The concept of a modern code paradigm will help us in parts 2 and 3 of this series of articles to articulate the facet of the broader EDVAC model that was implemented, for the first time, when ENIAC was converted to its new mode of operation. This coding scheme was not the only important innovation introduced with the First Draft in 1945, but it was certainly an important one and has perhaps been the most enduring.

Acknowledgments

This project was generously funded by Mrs. LD Rope's Second Charitable Settlement. Peggy Kidwell made a major contribution by alerting us to IBM's early internal use of "stored program," which Paul Lasewicz and Dawn Stanford were then kind enough to send to us from the IBM Corporate Archives. William Aspray, Atsushi Akera, Paul Ceruzzi, David Hemmendinger, and Martin Campbell-Kelly kindly answered our questions on specific topics and shared their perspectives on computing in the 1940s. Peter Sachs Collopy aided in the location and retrieval of archival material. This research benefited from inclusion in a special session organized by Gerard Alberts and Liesbeth De Mol for the 2013 Complexity in Europe Conference in Milan. Portions of this research were published earlier.[43]

References and Notes

1. D. Swade, "Inventing the User: EDSAC in Context," *The Computer Journal,* vol. 54, no. 1, 2011, pp. 143–147.
2. The First Draft was reprinted in J. von Neumann, "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing,* vol. 15, no. 4, 1993, pp. 27–75.
3. Some authors distinguish between "stored program concept" and "von Neumann architecture," attributing the latter to the much more specific computer design presented in A.W. Burks, H.H. Goldstine, and J. von Neumann *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument,* Inst. for Advanced Studies, 1946. The only serious historical claim that a computer storing its program in electronic memory was constructed without significant influence from the "First Draft" was made by Jack Copeland with respect to the Manchester Baby. J. Copeland, *Colossus: The First Electronic Computer,* Oxford Univ. Press, 2006, pp. 365, 373. After further research, Copeland has concluded instead that the design actually used for the Baby was derived directly from von Neumann's reports. B.J. Copeland, "The Manchester Computer: A Revised History Part 1: The Memory," *IEEE Annals of the History of Computing,* vol. 33, no. 1, 2011, pp. 4–21.
4. The three reports in the series are reprinted in W. Aspray and A.W. Burks, *Papers of John von Neumann on Computing and Computer Theory,* MIT Press and Tomash Publishers, 1987.
5. As we will discuss in "Engineering 'The Miracle of the ENIAC': Implementing the Modern Code Paradigm" in a forthcoming issue *of IEEE Annals,* the converted ENIAC also had real advantages over EDSAC in some areas, including maximum program length, ease of program writing and debugging, and input/output performance.
6. T.S. Kuhn, *The Structure of Scientific Revolutions,* Univ. of Chicago Press, 1962.
7. Von Neumann, "First Draft of a Report on the EDVAC," section 1.2.
8. Von Neumann, "First Draft of a Report on the EDVAC," section 2.5.
9. J.P. Eckert, Jr., "A Preview of a Digital Computing Machine," *The Moore School Lectures: Theory and Techniques for Design of Electronic Digital Computers,* M. Campbell-Kelly and M. R. Williams, eds., MIT Press, 1985, pp. 108–126. Quotations on pp. 112, 114.
10. J.W. Mauchly, "Preparation of Problems for EDVAC-Type Machines," *Proc. Symp. Large-Scale Digital Calculating Machinery, 7–10 January 1947,* W. Aspray, ed., MIT Press, 1985, pp. 203–207.
11. D.R. Hartree, *Calculating Instruments and Machines,* Univ. of Illinois Press, 1949, p. 88.
12. One can find different tube numbers quoted for many early computers, and in fact, the exact number would have fluctuated over their operating lives as hardware was added and removed. According to M.H. Weik, "BRL Report 971: A Survey of Domestic Digital Computing Systems," Aberdeen, 1955, ENIAC then had 17,468 tubes, the IAS computer around 3,000, and SEAC 1,424. S. Lavington, *Early British Computers,* Digital Press, 1980, reports "800 thermionic valves" for the Pilot ACE (p. 44), 3,000 for EDSAC, and as of April 1949, 1,300 for the

Manchester Mark 1 (p. 118). The new architecture was so effective in eliminating vacuum tubes that ENIAC's total was only ever exceeded by the immense AN/FSQ-7 computers that pushed the limits of 1950s computing technology for the military SAGE project.

13. Von Neumann, "First Draft of a Report on the EDVAC," section 5.6.
14. Campbell-Kelly and Williams, eds., *The Moore School Lectures.*
15. M.R. Williams and M. Campbell-Kelly, eds., *The Early British Computer Conferences,* CBI Reprints, MIT Press, 1985; D.R. Hartree *Calculating Machines: Recent and Prospective Developments and Their Impact on Mathematical Physics,* Cambridge Univ. Press, 1947; Hartree *Calculating Instruments and Machines;* Engineering Research Associates, *High-Speed Computing Devices,* McGraw-Hill, 1950.
16. N. Rochester, "A Calculator Using Electrostatic Storage and a Stored Program," 17 May 1949. From the IBM Corporate Archives, Somers, New York. Its system of two-digit instruction codes and three-digit addresses for the stored program was similar to the format adopted for the converted ENIAC.
17. J.W. Sheldon and L. Tatum, "The IBM Card-Programmed Electronic Calculator," *Papers and Discussions Presented at the Dec. 10–12, 1951, Joint AIEE-IRE Computer Conf.,* ACM, 1951, pp. 30–36, quotation on p. 35. However, the phrase "stored program" was previously used in print at least once prior to this to describe the MADDIDA as "controlled by a novel form of stored program" in an end-of-year summary of developments in "Electronic Computers" for an engineering audience. Anonymous, "Radio Progress During 1950," *Proc. IRE,* vol. 39, no. 4, 1951, pp. 359–396. There are almost certainly other early published uses waiting to be discovered.
18. W.H. Ware, *The History and Development of the Electronic Computer Project at the Institute for Advanced Study,* Rand Corp., 1953.
19. H.H. Goldstine, *The Computer from Pascal to von Neumann,* Princeton Univ. Press, 1972.
20. M.R. Williams, "A Preview of Things to Come: Some Remarks on the First Generation of Computers," *The First Computers: History and Architectures,* R. Rojas, and U. Hashagen, eds., MIT Press, 2000, pp. 1–16.
21. Comment by B. Randell, *Annals of the History of Computing,* vol. 3, no. 4, 1981, pp. 396–397.
22. M. Campbell-Kelly and W. Aspray, *Computer: A History of the Information Machine,* Basic Books, 1996, p. 104.
23. J. Copeland, "What Apple and Microsoft Owe to Turing,"12 Sept. 2013; www.huffingtonpost. com/jack-copeland/what-apple-and-microsoft-_b_3742114.html. One of us challenges this notion in T. Haigh, "Actually, Turing Did Not Invent the Computer," *Comm. ACM,* vol. 57, no. 1, 2014, pp. 36–41.
24. Wikipedia, "Stored-Program Computer," http://en.wikipedia.org/wiki/Stored-program_computer.
25. P. Ceruzzi, *Computing: A Concise History,* MIT Press, 2012.
26. D. Swade, "Inventing the User: EDSAC in Context," p. 146.
27. M. Priestley, *A Science of Operations: Machines, Logic, and the Invention of Programming,* Springer, 2011, argues that a connection between Turing's computational model and actual stored-program computers was only widely made after 1950.
28. A. Akera, *Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research,* MIT Press, 2007, p. 120, suggests that the stress placed by some historical writers on self-modifying code as a defining characteristic of the stored-program computer reflects theoretical concerns that were broadly articulated only in the 1960s.
29. The portion of the report dealing with programming was never written, but these functions were all present in ENIAC by 1945 and it is clear that the EDVAC team would have been concerned with preserving them. A contemporary report from the group noted that "von Neumann has specified that some order symbols be capable of modification by deleting a given part of the order and inserting something else in place of this part…. so that function tables may be used, subroutines called in, etc." J.P. Eckert, and J.W. Mauchly, "Automatic High Speed Computing: A Progress Report in the EDVAC. Report of Work Under Contract No. W_570_ORD_4926, Supplement No 4. (Plantiff Exhibit 3540),"30 Sept. 1945, ENIAC Patent Trial Collection, UPD 8.10, Univ. of Pennsylvania Archives and Records Center, p. 77. Von Neumann's sort program of 1945 uses instruction modification to, in later terminology, dereference a pointer. D.E. Knuth, "Von Neumann's First Computer Program," *ACM Computing Surveys,* vol. 2, no. 4, 1970, pp. 247–260.
30. This has been noted previously be several historians, including M.D. Godfrey and D.F. Hendry, "The Computer as von Neumann Planned It," *IEEE Annals of the History of Computing,* vol. 15, no. 1, 1993, pp. 11–21, and Priestley, *A Science of Operations.*
31. Priestley, *A Science of Operations,* pp. 167–169.
32. A.G. Bromley, *Stored Program Concept: The Origin of the Stored Program Concept,* tech. report 274, Basser Dept. of Computer Science, Univ. of

Sydney, Nov. 1985; cited at http://sydney.edu.au/engineering/it/research/tr/tr274.pdf. This ability is also essential to overlay or paging schemes. Bromley additionally suggested that self modification was necessary for compilers or assemblers to engage in "instruction building," but we do not believe this is correct in all cases.
33. A. Olley, "Existence Preceded Essence—Meaning of the Stored Program Concept," *History of Computing: Learning from the Past,* A. Tatnall, ed., Springer Verlag, 2010, pp. 169–178.
34. M. Bullynck and L. De Mol, "Setting-Up Early Computer Programs: D. H. Lehmer's ENIAC Computation," *Archive of Mathematical Logic,* vol. 49, 2010, pp. 123–146.
35. Priestley, *A Science of Operations,* chaps. 6 and 7.
36. The list has some overlap with the characteristics attributed to the stored-program concept in P. Ceruzzi, "Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935–1955" *IEEE Annals of the History of Computing,* vol. 19, no. 1, 1997, pp. 5–12, which shows that historians have invested "stored program" with a great deal more than the literal ability to store a program.
37. The order code specified in the report has been presented most clearly in Godfrey and Hendry, "The Computer as von Neumann Planned It."
38. One of the 10 arithmetic operations, *s,* would take a number from one or the other of the machine's arithmetic input registers depending on a flag set by the results of a previous arithmetic operation. Among other conditional operations, this could be used to set the address stored within an instruction to one of two possible values according to whether a particular condition was true or false. Von Neumann, "First Draft of a Report on the EDVAC," section 11.3.
39. M. Campbell-Kelly, "Foundations of Computer Programming in Britain (1945–1955)," doctoral thesis, Mathematics and Computer Studies, Sunderland Polytechnic, 1980, p. 239. We should note that our new evidence shows ENIAC had already run a complex Monte Carlo program in the modern code paradigm prior to this date.
40. M. Campbell-Kelly, "Programming the Pilot ACE: Early Programming Activity at the National Physical Laboratory" *Annals of the History of Computing,* vol. 3, no. 2, 1981, p. 138.
41. Burks, Goldstine, and von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.*
42. M.V. Wilkes, D.J. Wheeler, and S. Gill, *The Preparation of Programs for an Electronic Digital Computer,* Addison-Wesley, 1951.
43. T. Haigh, " 'Stored Program Concept' Considered Harmful: History and Historiography," *The Nature of Computation. Logic, Algorithms, Applications,* P. Bonizzoni, V. Brattka, and B. Löwe, eds., LNCS 7921, Springer-Verlag, 2013, pp. 241–251.

Thomas Haigh is an associate professor of information studies at the University of Wisconsin–Milwaukee. His research interests include the history of computing, especially from the viewpoints of labor history, history of technology, and business history. Haigh has a PhD in the history and sociology of science from the University of Pennsylvania. See more at www.tomandmaria.com/tom. Contact him at thaigh@computer.org.

Mark Priestley is an independent researcher into the history and philosophy of computing, with a special interest in the development of programming. He started his career as a programmer and was for many years a lecturer in software engineering at the University of Westminster before turning to the history of computing. Priestley has a PhD in science and technology studies from University College London. His most recent book, *A Science of Operations: Machines, Logic, and the Invention of Programming* (Springer, 2011), explores the coevolution of programming methods and machine architecture. More information is available at http://www.markpriestley.net. Contact him at m.priestley@gmail.com.

Crispin Rope has been interested in ENIAC since reading Douglas Hartree's pamphlet on the machine from 1947 and has pursued an avocational interest in its history for more than a decade. His earlier work on this topic has been published in *IEEE Annals of the History of Computing* and *Resurrection: The Bulletin of the Computer Conservation Society.* Contact him at westerfield@btconnect.com.