‘Stored Program Concept' Considered Harmful:
History and Historiography*

Preprint version. Final version published in *The Nature of Computation. Logic, Algorithms, Applications,* ed. Paola Bonizzoni, Vasco Brattka, Benedikt Löwe, Springer 2013 (LNCS 7921). http://link.springer.com/book/10.1007/978-3-642-39053-1.

Thomas Haigh

University of Wisconsin—Milwaukee, Milwaukee, United States

thaigh@computer.org

Abstract. Historians agree that the stored program concept was formulated in 1945 and that its adoption was the most important single step in the development of modern computing. But the “concept” has never been properly defined, and its complex history has left it overloaded with different meanings. The paper surveys its use and development and attempts to separate it into three distinct aspects, each with its own history and each amenable to more precise definition.

Keywords: ENIAC, von Neumann Architecture, Turing Machine, History of Computing.

1 Introduction

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. Historians also agree that the concept was first stated in the “First Draft of a Report on the EDVAC,” (hereafter “First Draft”) circulated under the name of John von Neumann in 1945 [1]. While the true balance of credit for the ideas contained in this document is widely and heatedly debated, its fundamental influence on the development of modern computing is not.

Yet when historian Doron Swade delivered an address [2] to celebrate the sixtieth anniversary of the 1949 EDSAC computer, generally considered the first useful stored

* This paper draws extensively on ideas and analysis developed during my ongoing collaboration with Mark Priestley and Crispin Rope on a project exploring the ENIAC’s 1948 conversion to a new programming method and its use for the first computerized Monte Carlo calculations. In particular the definitions given of the “modern programming paradigm” were formulated during discussion with Priestley.

program machine, he began with the startling observation we do not really agree on why the concept is so important. For years Swade had "assumed that the significance of the stored program must be self-evident" and attributed his own confusion to personal inadequacy. Eventually he "became bold and began asking" computer historians and pioneers to explain it. 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.

The problem, I think, is that we have never actually agreed that the "stored program concept" is. The concept is sometimes treated as an approach to programming, sometimes as a new kind of architecture. Some authors conflate it with the idea of a universal machine and associate it with Turing's ideas on computability. Sometimes the idea is defined very narrowly, with attention to the interchangeable storage of programs and data, and sometimes as a grand cluster of ideas accumulated over time. Self-modifying code may or may not loom large in the discussion. Some authors use "stored program concept" and "von Neumann architecture" interchangeably, while others attempt to separate them.

As a historian, my instinct is to explain this proliferation of meanings and associations historically, going back to the conventional origin point of the stored program concept in 1945 and sketching its subsequent development in the hands of different groups of people with different intellectual agendas.

My own attention was drawn to the concept of "stored program" as we investigated modifications made to ENIAC in early 1948 by a team working closely with John von Neumann. These changes incorporated key elements of the stored-program approach several months before Manchester University's "Baby" computer executed what is usually called the world's first stored program in the summer of the same year. Confusion of the kind noted by Swade makes it impossible to clarify the status of the converted ENIAC, and several other important early machines, as "stored program" or "not stored program."

The final part of the paper attempts to begin the work of separating the cluster of ideas treated as part of the stored program concept by identifying three distinct paradigms stated in von Neumann's seminal 1945 report that were incorporated as standard features by most machines of the 1950s.

2 History of the Stored Program Concept

2.1 The 1945 EDVAC Report

The "First Draft of a Report on the EDVAC" does not, despite the role it has been assigned in later historical work, function very well as a standards document to rigor-

ously define the concept of "stored program." Most notably the word "program" never appears. Von Neumann consistently preferred "code" to "program" and wrote of "memory" rather than "storage."

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 medium. The First Draft itself observed that "instructions must be given in some form which the device can sense: Punched into a system of punchcards 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." [1].

The First Draft argued for the collocation of code and data, though only tentatively: "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…."[1]. 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, in comparison, be quite small. Using one set of mechanisms to manipulate both would simplify EDVAC.

2.2 Initial Reception

The First Draft circulated quickly between those interested in building computers, and almost immediately established the model for the next generation of computer project. However, early discussion of the EDVAC approach addressed a range of innovative features, with no consensus that the particular method of program storage was the most important one. Several early introductory computing books focused primarily on the method of programming, documenting the instruction set of von Neumann's planned computer at the Institute for Advanced Studies.

To many of the computer builders of the 1940s, including ENIAC creators J. Persper Eckert and John Mauchley whose contribution to the new design was profound, the advantage of what is sometimes called the "EDVAC-type" approach was seen primarily as a way of building a powerful and flexible computer with a relatively small number of expensive and unreliable vacuum tubes. Creation of a large high-speed memory was the key engineering challenge posed by this approach. Ease of programming and speed of changeover from one problem to another were acknowledged as benefits of this approach, and were related to the storage of code and data in the same electronic memory.

2.3 The Phrase "Stored Program"

We could not find the phrase "stored program in any of the early computing conference proceedings and primers published in the 1940s. Its earliest known usage occurred in 1949 by a small team at IBM's Poughkeepsie facility producing the "Test Assembly," IBM's first EDVAC-type computer. This experimental system was built

around firm's first electronic calculator, its 604 Electronic Calculating Punch, which became the arithmetic unit of the lashed-up computer. They added a new control unit, cathode ray tube memory, and magnetic drum.

An internal proposal written by Nathaniel Rochester in 1949 [3] noted that the plug-board approach was not viable with large programs, which could be solved by reading "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." To distinguish between the program held on the 604's standard plug board and the new-style programs stored either in the 250 word electronic memory or on the drum the team began to call the latter the "stored program." Rochester's document was titled "A Calculator Using Electrostatic Storage and a Stored Program." (Thanks go to Peggy Kidwell of the Smithsonian for alerting me to this document).

Within IBM the meaning of "stored program" quickly evolved from a literal description of a particular kind of programming mechanism into a general description of EDVAC-type machines. During the 1950s the phrase pops up occasionally in conference papers, particularly those delivered by IBM employees, but as all powerful digital computers by then followed this model people generally just referred to "large-scale digital computer."

2.4 Historians Adopt "Stored Program"

In the late 1970s and 1980s the history of computing emerges as an academic subfield. Early work focused on the machines of the 1940s. Following the lead of pioneer turned historian Herman Goldstine the idea of the "stored program computer" was borrowed from technical discourse, developing from a fairly obscure term into a central concept in heated debates over what should be considered the first true computer and why.

Put simply, the community resolved this intractable and distracting wrangle by agreeing on a string of words that clarified what each of the key early machines had accomplished. ENIAC, for example, became the first "large-scale general-purpose digital electronic computer to be fully operational." The Cambridge EDSAC and Manchester Baby were recognized as the first operational stored program computers, patterned after the EDVAC design, but there was little need or interest in defining the "concept" more rigorously to identify its necessary and sufficient characteristics.

More recently, historians of computing largely turned their attention away from the 1940s, having reached a consensus on the honors to be granted to each early machine and formulated a consensus narrative on the events of the decade. As Swade noted, the stored program concept is still enshrined in popular and scholarly histories as a key transition but receives little new analysis.

2.5 The Stored Program Concept Meets Universality

From the late 1950s onward, with the rise of theoretical computer science and a new enthusiasm for work on abstract models of computation, the digital computer was increasingly reinterpreted as an embodiment of the universal Turing machine. Its key

benefit was therefore the ability of the computer to treat instructions as data and modify them programmatically. The confusion encountered by Swade seems to reflect differences in opinion held by those influenced by the pragmatism of the 1940s and those favoring the theoretical concerns developed later. 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. In turn, the concepts of stored program and general purpose computer have sometimes conflated with the more formal concept a computer being Turing complete or "universal" if equipped with a memory of infinite size.

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… the stored program computer idea can be traced back to the 1936 theoretical concept of a universal Turing machine." In his recent *Computing: A Concise History* [4], 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." Even Swade himself retreated from the endearingly bold confession of confusion quoted earlier to the rather conventional conclusion 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." [2]

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 very few references to Turing's theoretical work among the discussions of those building computers in the 1940s [5, 6]. Atsushi Akera [7] has 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. In later discussion the advantages of stored program machines were often justified according to the theoretical concerns of later years rather than the pragmatic issues of primary importance to their designers.

In this sense, the search for the logical foundations of computing and the search for is historical foundations may pull us in opposing directions. It was its very purity and abstraction from the messy details of actual hardware that earned the Turing Machine its iconic place within computer science. The ideas of Turing completeness and of the Universal Turing Machine served to decouple theoretical computer science from the material world of computing platforms and architectures. For example, once a virtual computer built within Conway's Game of Life was shown to be computationally equivalent to a Universal Turing Machine that single fact told us that, with sufficient

time and a large enough cellular matrix, this computer could execute the same algorithms as any machine build from conventional components. The intellectual utility of this approach is clear, as is its strategic benefit to the early computer science community at a time in which it was struggling to separate itself intellectually from mathematics, electronic engineering, and scientific service work to other disciplines.

The late Michael Mahoney struggled for many years to encapsulate the history of theoretical computer science [8]. His great theme was the need of scientific communities to construct their own historical narratives. Mahoney saw theoretical computer science as an assemblage of mathematical tools originally developed in quite separate contexts, from group theory and Lambda calculus through to Chomsky's hierarchy of grammars. On a still larger scale, work on mathematical logic and on the engineering of calculating machines both had long but largely distinct histories. Yet from within the discipline and from the present-day viewpoint the connections between these things came to seem obvious and history is often written as if work over the centuries had been directed towards the development of the computer or as if the computing pioneers of the 1940s were inspired primarily by the work of Turing.

As Mahoney wrote, the interest of practitioners in "finding a history… has its real dangers" because while scholarly historians and practitioners "both seek a history" it is "not for the same purpose and not from the same standpoint." [8]Abstraction is the soul of computer science, but as historians we lose something vital if we abstract away from the historical grubbiness of early computer projects, their focus on engineering challenges, their specific goals and roots in the thinking of the 1940s. The abstraction from real computers and real computing practice provide by a focus on Turing completeness is good for theory but bad for history. For example, the effort by Raul Rojas to claim Konrad Zuse's 1943 Z3 computer as universal [9] is an impressive party trick, but diverges entirely from the way in which the machine was designed, how it was actually used, or indeed from anything that would have made sense in the 1940s. The programming method described construction of an impossibly long paper tape, and a massive loss of computational performance. Calculation would have been quicker by hand. To me the real lesson is that the Z3 could have been Turing complete with only minor design changes, but wasn't because the concept and its benefits were not yet widely understood. Indeed, Zuse later claimed to have considered and rejected treating program instructions as data while working on its design. The past really is a foreign country. Yet Rojas was able to raise the status of the machine, and German pride, with this appeal to the world of theory.

In this context, it is worth noting that von Neumann's 1945 report specifically forbade unrestricted code modification, a notable conceptual divergence from what is now understood as the Turing machine model of the universal computer. The EDVAC described therein did rely on code modification for many common operations, including loop termination, other kinds of conditional branching, and altering the address data is fetched from (for example to obtain a value from a different cell within an array each time a block of code is looped through). This reflected a broader design philosophy of radically simplifying computer architecture by replacing the special purpose mechanisms common in earlier designs with a small number of general purpose mechanisms.

However, von Neumann's instruction set for EDVAC explicitly prevented instructions from being fully overwritten [5, 10]. Only address fields could be changed. One of the 32 bits in each word of memory flagged it as holding either program or data. A transfer operation applied to an instruction word would overwrite only the address field.

3 Beyond "Stored Program"

Like the Goto statement, discussion of the "stored program concept" has outlived the purpose for which it was created and provides a shortcut to confusion. The time has come to replace it, as an analytical category, with a set of more specific alternatives amenable to clear and precise definition. Below I discuss one proposed partial replacement in some detail and sketch two more.

3.1 The Modern Code Paradigm

The first of these is the "modern code paradigm." This new term describes 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 (such as the lack of a dedicated conditional branch instruction), while some common code capabilities of 1950s computers (such as index registers) came from other sources.

Looking for novel code-related features from the 1945 First Draft that had become taken-for-granted features of computers a decade later illuminates the process by which a sprawling, idiosyncratic and brilliant document became a dominant paradigm for the builders of computers.

As one reviewer of this paper noted, this analysis parallels the work of computing theorists to build abstract models of computation based around the stored program approach rather than Turing machines. These include the Random Access Machine (RAM) and Random Access Stored Program machine (RASM). Space does not permit further comment, except to point out that the objectives of the historian and the theorist remain distinct. Whereas the theorist looks for the minimal necessary capabilities for universality, my objective here is to define the maximal set of features present in the 1945 draft that actually made it into the standard designs of the 1950s.

1. **The program is executed completely automatically.** To quote the First Draft, "Once 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 manual intervention at branch points had been workable with slower devices such as the Harvard Mark I.
2. **The program is written as a single sequence of instructions, known as "orders" in the First Draft, which are 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 did specify the explicit demarcation of memory locations holding

code from those holding data. 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."

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. Altogether, orders required between 9 and 22 bits to express. Actual machines usually followed this pattern. 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."
5. **However, a program can instruct the computer to depart from this ordinary sequence and jump to a different point in the program.** "There must, however, be orders available which may be used at the exceptional occasions referred to, to instruct CC to transfer its connection [i.e. fetch the next instruction from] any other desired point" in 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 thirteen digits, representing the address μρ, should be overwritten. Actual computers achieved functionally equivalent capability through some combination of unrestricted code modification, indirect addressing mechanisms, and conditional branch instructions.

A consequence of the above 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 a variety of resources such as program lines, plug board capacity, or tape readers as potential limitations on logical program complexity.

3.2 The von Neumann Architecture

The modern code paradigm is not intended 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 former 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, one of these facets 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, so that only one operation takes place at a time, and the routing of all memory transfers through the central arithmetic unit. Also the system of special purpose registers to serve as source and destination for arithmetic and logic instructions, and to provide a program counter and instruction register for control purposes. The von Neumann architecture has general been more clearly defined within the technical literature than has the stored program concept. One might, as several have, dispute the extent to which it is fair to attach only von Neumann's name to these concepts. "EDVAC architecture paradigm" could serve as an alternative.

3.3 The EDVAC Hardware Paradigm

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" report include 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). With the possible exception of the memory technologies discussed these hardware features were not unique, but collectively they represented a bold commitment to new technologies at a time when computing group 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 Thomas Kuhn's core sense of a powerful and tangible exemplar [11].

3.4 Separate Trajectories

These three paradigms have intertwined early histories, but were always at least partially separable and ultimately diverged. Many machines of the 1940s implemented some aspects of the EDVAC paradigms but not others. 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 12th May 1948 would make this "the first operational EDVAC-type stored program computer (although it was not of course electronic)." [12] Alan Turing's design for the ACE adopted von Neumann's architecture and fol-

lowed 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 because its architecture was quite unlike that of any modern computer…." [13] ENIAC after its 1948 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.

The machines of the mid-1950s tended to implement all three paradigmatic aspects of the First Draft's design for EDVAC. The paradigmatic influences of these three were diverging again by the end of the decade. Its relevance 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, though its primacy was gradually chipped away as 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 as a description of the machine language executed by processors (though not of the languages used by humans to write programs). It was extended and made more specific in many ways, not least by von Neumann's own 1946 description of the planned structure of his Institute for Advanced Studies machine. [14] It was not, however, overturned.

References

1. von Neumann, J.: First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing 15, 27-75 (1993)
2. Swade, D.: Inventing the User: EDSAC in Context. The Computer Journal 54, 143-147 (2011)
3. Rochester, N.: A Calculator Using Electrostatic Storage and a Stored Program. IBM's Early Computers Sources collection. IBM Archives (1949)
4. Ceruzzi, P.: Computing: A Concise History. MIT Press, Cambridge, MA (2012)
5. Priestley, M.: A Science of Operations: Machines, Logic, and the Invention of Programming. Springer, New York (2011)
6. Lavington, S. (ed.): Alan Turing and his Contemporaries. British Informatics Society Ltd, Swindon, UK (2012)
7. Akera, A.: Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research. MIT Press, Cambridge, MA (2006)
8. Mahoney, M.S., Haigh, T. (ed.).: Histories of Computing. Harvard University Press, Cambridge, MA (2011)
9. Rojas, R.: How to Make Zuse's Z3 a Universal Computer. IEEE Annals of the History of Computing 20, 51-54 (1998)
10. Godfrey, M.D., Hendry, D.F.: The Computer as von Neumann Planned It. IEEE Annals of the History of Computing 15, 11-21 (1993)
11. Kuhn, T.S.: Second Thoughts on Paradigms. The Essential Tension: Selected Studies in Scientific Tradition and Change, pp. 293-319. University of Chicago Press, Chicago (1979)

12. Campbell-Kelly, M.: Foundations of Computer Programming in Britain (1945-1955). Ph.D. thesis, Mathematics and Computer Studies, Sunderland Polytechnic (1980)
13. Campbell-Kelly, M.: Programming the Pilot Ace: Early Programming Activity at the National Physical Laboratory. Annals of the History of Computing 3, 133-162 (1981)
14. Burks, A.W., Goldstine, H.H., Neumann, J.v.: Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Institute for Advanced Studies, Princeton, NJ (1946)