DOI:10.1145/2846088

Thomas Haigh and Mark Priestley

Historical Reflections

Where Code Comes From: Architectures of Automatic Control from Babbage to Algol

Considering the evolving concept of programming.

In our previous *Communications* column (September 2015) we noted that a celebrated table published by Ada Lovelace in 1843 was not a computer program, despite frequent claims to the contrary. Here we turn to a related question: where did computer code come from? Back in the 1840s nobody talked about "programming" Charles Babbage's planned engines. More importantly, nobody had yet formulated the concept of a program as a series of instructions controlling the operation of a general-purpose computer. The work of Charles Babbage and Ada Lovelace provides an important milestone on the road to this invention, but marks the beginning of the story rather than its end.

In this column we explore the rest of that story, returning briefly to the world of Lovelace and Babbage before moving on to the 1940s when their ideas were independently rediscovered, extended, and finally realized in actual machinery. Developments came thick and fast, moving in just a few years from the earliest relay computers controlled by "coded" arithmetic instructions on tape to ENIAC, the first computer to automatically carry out computations with complex structures including branches and nested loops. This was the context in which the word "programming" was initially applied to a computer, originally to describe the action of the machine's control wires, circuits, and switches when triggering the appropriate sequence of mathematical operations. Before ENIAC was even finished its creators, in collaboration with John von Neumann, had come up with a new approach in which control and arithmetic operations were both represented in a single series of coded instructions stored in an addressable memory unit. That was soon called a computer program, and although the meaning of the term has continued to evolve it has retained this basic sense of a set of instructions that direct the performance of a series of operations, enabling a computer to carry out a task without human intervention.

U.S. ARMY PHOTO

U.S. Army photograph of the installed EDVAC.

Babbage Proposes a Control Operation

Unlike Babbage's earlier proposed calculator, the Difference Engine, the Analytical Engine was what would later be called a "general-purpose" automatic computer. This flexibility meant it would have to be "ordered" or "instructed" to carry out whatever particular sequence

of arithmetic operations (addition, subtraction, multiplication, or division) was needed. These orders were punched onto "operation cards" joined in a continuous chain, forming something analogous to the control roll of a player piano or the paper tapes used by later computers.[a] A separate sequence of "variable cards" told the Engine which locations in its "store" to use for the arguments and results of each operation.

Babbage recognized that punching a card for every operation needed in a complex calculation would be inefficient and inflexible, as most computations have a structure in which sequences of operations are repeated. Fully automating computation meant making explicit and mechanizing not just sequences of arithmetic operations but also the control processes needed to direct these repetitions. Babbage envisioned mechanisms to "back up" the chains of cards to repeat a sequence of operations. The rewinding operation would be triggered by special "combinatorial cards" placed among the operation cards. When Lovelace published details of her planned computation, in what has become one of history's best-known endnotes, she relied on this capability when defining two nested loops. In our previous column we observed that her table omitted the control information needed to direct the Engine through these loops and was closer to being a simulated trace than an actual program. She described the overall structure of the computation more abstractly in a symbolic expression inspired by mathematical notation.

This omission is not surprising, as Babbage's ideas about the backing-up scheme were still rather provisional. Only in 1844 did he even prepare lists of the operations that his engine would support.[1] Babbage specialist Allen Bromley described them as documenting a "programmer's interface." Along with the expected arithmetic operations, Babbage defined an operation of "Ascertaining if any Variable = 0." The card ordering this operation would specify how many cards were to be skipped if the variable defined by the current variable card was zero. That marked a crucial generalization of the notion of "operation" beyond the familiar operations of arithmetic to encompass the control operations that determine exactly which sequences of arithmetic operations are carried out.

Punched Cards and Analog Computers

The Analytical Engine was never built, or even completely and stably designed. Over the next nine decades a variety of calculating and counting machines were developed, including various kinds of punched card tabulating equipment and a number of differential analyzers. None of their designers attempted to revive Babbage and Lovelace's pursuit of a general-purpose automatic computing machine. Configuring these machines was not called "programming."

The punched card machines each tackled a specialized operation, such as tabulating cards or sorting them. Each could be wired to carry out a particular variant on this task, for example to ignore some columns on the card while calculating totals and subtotals based on others. While these machines were applied to scientific calculations from the 1930s onward most of the work we think of as executing a program was carried out by human operators, not by the machines themselves.

Analog computers, such as differential analyzers and gun-control systems, took a fundamentally different approach, representing numbers not as digits but as continuously varying quantities. Changing variables were modeled as changes in voltage or mechanical rotation in a particular unit within the machine. The computers could be configured, often with wrenches or screwdrivers, to specify particular relationships between these units, modeling the terms in an equation. Each piece of the machine performed a single task throughout the computation. These machines were not following instructions as they computed, and so the concept of a program, according to which devices carry out a sequence of different operations over time, does not apply to them.

Codes and Coding

We shall therefore jump forward from the time of Babbage directly to the early 1940s, when the first automatic digital computers were being built. Their capabilities were similar to those Babbage planned for the Analytical Engine, though they were conceived without knowledge of his designs and most used electromechanical relays rather than cogwheels to represent numbers. They included the Mark I computer built by IBM for Harvard University, a series of wartime machines built by Bell Labs in New York, and the Z3 built by Konrad Zuse in Berlin.

Like the planned Analytical Engine, these machines carried out sequences of arithmetic operations, now represented as patterns of holes punched in control tapes. These patterns were called "codes," mirroring earlier uses of "code" to describe the encoding of messages onto paper tape for machine transmission (Baudot code) or into dots and dashes for human transmission (Morse code).[b]

The designers of Mark I thought of these orders as primarily arithmetical, each specifying an operation to carry out as data was transferred from one register to another. Rather than winding the control tape back to repeat sequences, loops were implemented in the most literal sense possible: gluing the ends of the control tape together to form a physical loop. This is, we strongly suspect, the origin of the term "loop." Mark I thus used coded orders for arithmetic, but not for control structures. Loops were mapped onto the physical configuration of the tape, and transfers of control were carried out manually by humans. It took code, paper loops, and humans to carry out the functions later automated with program code alone, so specifying a problem to run on Mark I required preparation of both coded orders for the machine and detailed instructions for its human operators.

A small number of orders did perform control functions, most impor-

a Charles Babbage, "On the Mathematical Powers of the Calculating Engine," 1837 manuscript reprinted in several collections.

b For example, in the Bell Labs case "the numerical results may be translated into special codes and perforated on standard teletype tape." (B.L. Sarahan, "The Relay Interpolator: A Description of its Operation," Naval Research Laboratory Report R-3177, Sept. 25, 1947, v.) Likewise, in Mark I "the perforations in the control tape corresponding to code 21 in the Out column." (Staff of the Harvard Computation Laboratory, *A Manual of Operation for the Automatic Sequence Controlled Calculator*. Harvard University Press, Cambridge, MA, 1946, 14).

tantly one that halted the machine automatically when a loop termination condition was satisfied. At this point, a machine operator would remove the looped control tape from the tape reader and replace it with whatever sequence came next. Changing tapes manually might seem inefficient, but the machine ran so slowly that operators could plausibly keep up with them. Loop execution times on Mark I were measured in minutes rather than microseconds.

The words "program" and "programming" were not originally applied to these machines. However, by 1944 the staff of the Harvard Computing Laboratory had recognized the work of "coding" problems into sequences of operation codes as a distinct task: "... the mathematician ... chooses the numerical method ... such functional, value and control tapes as are required are then computed, coded and punched. Since the mathematician cannot always be present while the calculator is running, instructions must be prepared to guide the operating staff."[c]

The system of tape-driven automatic control was later extended by providing instructions to shift control between more than one tape reader. For example, a computer with four readers might use them for code sequences corresponding to inner loop, outer loop, initial setup, and the printing of results. This system was stretched to breaking point, and beyond, with the completion in 1948 of IBM's Selective Sequence Electronic Calculator. As the word "Selective" suggests, the SSEC could automatically select which of several dozen paper-tape readers to take its next instruction from. SSEC staff had to grapple with tape rolls weighing 400 pounds, used to prepare data tapes looped at high speed past multiple read heads so that values could be looked up from coded tables. A custom lift was engineered to hoist these tapes, which were so wide that a special machine was built to glue their ends together.

ENIAC and the Automation of Control

The term "programming" comes, a little indirectly, from the project to build a much faster electronic computer at the University of Pennsylvania. The unprecedented speed of ENIAC, completed in 1945, forced its designers to come up with an entirely different control system. No paper tape could possibly read operation codes rapidly enough to keep its electronic circuits busy. Neither was it practical to expect operators to change tapes every few seconds as ENIAC completed a loop or subroutine and needed to move on to the next sequence of operations.

The words "program" and "programming" were not originally applied to these machines.

People often expect the history of technology to consist of a fairly direct series of advances by which primitive old machines gradually come to look and act ever more like modern ones. ENIAC is difficult to fit into this view of history. It was the first general-purpose electronic digital computer, being reconfigured to tackle entirely different kinds of problems from weather forecasting to prime number detection. Its control mechanism provided the full range of capabilities we associate with modern computers, including conditional branches and nested loops, but used an entirely different approach.

As we explain in our new book, *ENIAC in Action: Making and Remaking the Modern Computer*,[3] ENIAC consisted of dozens of distinct units, most built to carry out specialized computational functions such as multiplication, addition and number storage, loop control, or table look-up. When one unit finished a task it generated a "program pulse" to inform the unit responsible for the next operation that it was time for it to wake up and do something. What ENIAC did next was determined by two things. The first was its wiring, as the destination of the program pulse depended on where in ENIAC the other end of the wire carrying it had been plugged. The second factor was the switch settings on the receiving unit. ENIAC resembled a RadioShack boxed electronic kit, in that configuring it for a particular job involved wiring together the units needed to build a special-purpose machine.

The ENIAC team initially called the task of configuring ENIAC to carry out a particular problem "setting up" the machine. A particular configuration was called a "set-up" and documented in a diagram showing the wiring patterns and switch settings needed on each unit. This representation is quite different from a modern program, or even from the Harvard Mark I control tape. Indeed, the ENIAC approach to sequencing operations is much more difficult for modern audiences to grasp than the coded instructions used by the relay computers.

When computer scientists look back at the computers of the 1940s it is often to argue about which of them were "Turing complete." This depends in large part on their ability to implement a conditional branch, meaning the ability to select between two possible courses of action. Deciding whether to terminate a loop is seen as a special case of conditional branching, which is indeed how the instruction sets of later computers implemented looping.

Babbage, Lovelace, and the designers of ENIAC, however, modeled the top-level structure of computations in terms of loops rather than branches. Lovelace's mathematical notation expressed a computation in summary form as nested sequences of operations repeated a certain number of times. ENIAC's designers first considered building "about 30 units which are capable of receiving program pulses on one line and transmitting them on either of two lines in accordance with pulses received on another line." If one conceptualizes ENIAC's control wires as rails and its control pulses as trains, these switches would steer the pulses onto one or another track depending on the control signals received. This is perhaps why the term "branch" was later introduced to describe this control action, as pulses literally followed one or another branch through ENIAC's networks of control wires depending on the operations of its control circuits.

However, ENIAC's designers soon rejected simple binary switches, in favor of more complex "steppers"

c Ibid, p. 50.

each able to trigger up to six different sequences of operations. ENIAC's "master programmer" unit combined enough steppers and counters to count the iterations of each sequence and control up to 10 nested loops. Routing a control signal to a special input on each stepper would terminate a loop immediately, meaning that looping mechanisms also supported simple conditional branching. It was as if the two alternative statements in an "if ... then" statement were treated as loops that would be iterated at most once. Inverting the later conception of looping as a special case of conditional branching, ENIAC made looping and loop termination the fundamental behavior.

Early in the ENIAC project, before design work had progressed very far, philosopher turned engineer Arthur Burks produced a table showing how ENIAC could compute an artillery trajectory, the task for which the machine had been commissioned. Although independently developed, the structure of Burks' table strongly resembled that produced a century earlier by Lovelace. In both tables, rows represented steps in the calculation, each storage unit was given its own column, and cells showed the content of each unit at each point in the calculation.[d] As we discussed in our September 2015 column, the tables are not in themselves programs, and are best viewed as traces or walkthroughs of the machine's operation.[e] Both tables indicated a need for nested loops, but when they were produced neither target machine had a well-defined mechanism for iteration. In a sense, the tables served as functional specifications for the machine designers: devise a mechanism to generate this sequence of operations and your machine will successfully complete this computation.

This striking convergent evolution, despite the very different architectures of the two machines, shows the analysis of a problem and its reduction to a series of arithmetic operations had very little to do with the specifics of the control system that would ultimately direct those operations. Indeed, the methods used to plan computations for automatic computers often incorporated those used with earlier technologies, whether in the application of punched card machines or desk calculators to large-scale mathematical work or the analysis of printed forms and clerical procedures in the office.

Earliest Discussion of "Programming"

Our reference to ENIAC's "master programmer" in the previous section alerts you to two things. The first is the word "program" became entangled with the control of automatic computers during the ENIAC project.[f] The second is it did not mean what you expect. By the 1950s "master programmer" would read as a slightly odd job title. In 1944 it was a pair of boxes stuffed with electronics to repeatedly trigger sequences of operations by generating control pulses. In fact the words "program" and "programming" cropped up in project documents to describe many different aspects of ENIAC's control system. As well as calling its control signals "program pulses," a June 1944 progress report described two accumulator units as being "automatically programmed to receive the multiplier and multiplicand" when a program pulse triggered the multiplier unit to which they were attached. This use of "program" fits with the notion, familiar to Babbage, that an automatic computer is built to carry out defined sequences of operations. Its control mechanism must trigger the performance of the right operations in the correct order. This is very similar to the meaning of "program" in other contexts—for example, the work of a radio programmer who selects and schedules programs for broadcast, the program for a series of concerts, or the program of study followed over time by a student. The use of "programmer" as the name for a simple mechanical control unit on a washing machine reflects a similar usage—turning the dial to a particular point triggers the performance of a particular sequence of washing operations (spin, rinse, wash, and so on). Echoing this, a primary meaning of "program" on the ENIAC was to describe a single operation set up on one of its units. What were being programmed were the operations of the internal circuitry of that unit.

By late 1945, however, the ENIAC team was beginning to talk of "programming" in something much closer to its modern meaning. This reflected the emergence of an entirely new way to think about automatic control.

EDVAC and the Modern Code Paradigm

"The First Draft of a Report on the EDVAC," composed in the spring of 1945 by mathematician John von Neumann and based on his work with members of the ENIAC team, never led to a second draft, still less a published article. It nevertheless laid out the basic architecture from which almost all subsequent computers have evolved. Computers patterned after the basic structure von Neumann proposed for the EDVAC, a successor to ENIAC being designed at Penn under a government contract, are often called "stored program" computers. We have previously criticized this term as vague and irredeemably overloaded with conflicting meanings, but those words do at least have the virtue of suggesting the attractiveness of EDVAC had something to do with its control system.[5]

EDVAC, as described by von Neumann, would drop ENIAC's special-purpose units and its elaborate system of distributed control. Like Babbage's Analytical Engine and the relay computers of the 1940s, EDVAC would read and decode orders one at a time, performing the operation specified by the code. The novelty was the code integrated control and arithmetic instructions in a single, aggressively minimalistic, set of orders. EDVAC did not need the hybrid control schemes of the relay machines or the special-purpose mechanisms and programming wires and switches of ENIAC.

We have previously identified the key aspects of the EDVAC approach to automatic control as:

- The program is executed completely automatically.
- The program is written as a single sequence of instructions, known as

d We have exploited this similarity to produce an ENIAC set-up that performs the Bernoulli calculation as specified by Lovelace. Run on an ENIAC simulator, it does indeed generate the sequence of Bernoulli numbers.

e Allen Bromley used the term "walkthrough" to describe tables like Lovelace's in "Charles Babbage's Analytical Engine, 1838," *Annals of the History of Computing 4*, 3 (1982), 215.

f Discussed in D.A. Grier, "The ENIAC, the verb "to program" and the emergence of digital computers." *IEEE Annals of the History of Computing 18*, 1 (Jan. 1996), 51–55.

"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.

- Each instruction within the program specifies one of a set of atomic operations made available to the programmer.
- The program's instructions are usually executed in a predetermined sequence.
- However, a program can instruct the computer to depart from this ordinary sequence and jump to a different point in the program.
- The address on which an instruction acts can change during the course of the program's execution.[5]

Von Neumann's design melded facilities for arithmetic and control. It contained both types of instruction, similarly formatted. His arithmetic circuits could be used to conditionally select numbers, while his storage circuits could change destinations for jump instructions as well as overwriting numeric data. This unification of control and arithmetic operations was more important than, and facilitated, EDVAC's more celebrated innovation of storing both instructions and data in the same writable and addressable memory. The first known program written in the EDVAC style was developed by von Neumann himself, and is now on display at the American Philosophical Society in Philadelphia.[6] The first to be run on an actual computer was executed directly from a read-only memory on ENIAC in April 1948, after its conversion to the new programming mode.[4] A few months later, at the University of Manchester, a program was loaded into an experimental writable memory and executed.

In the First Draft, von Neumann followed the Mark I terminology, giving an "order code" that defined EDVAC's instruction set. This usage was extended in an influential series of reports from the computing team he set up at the Institute for Advanced Studies in 1946 to construct his own EDVAC-like computer. These reports divided the process of problem preparation into two broad phases. "Planning" was described as "a mathematical stage of preparations," but "coding" encompassed drawing flow diagrams as well as writing instructions.[g]

At Penn, however, the meaning of the verb "to program" quickly shifted from describing the action of the control circuits responsible for triggering operations at the correct time to describing the work of the humans devising such sequences. In late 1945, a report described the practices used in "planning a set-up for the ENIAC" as "programming techniques,"[2] and a letter from one of the project's leaders noted "the EDVAC will contain a large number of units capable of remembering programming instructions," to be copied from tape "before the actual program is started."[h] "Programming" was by then roughly synonymous with von Neumann's "coding," and by early

g Goldstine, H.H. and von Neumann, J. *Planning and Coding Problems for an Electronic Computing Instrument, Part II, Volume 1* (Apr. 1, 1947, section 7.9). Drawing flow diagrams was described as the "dynamic or macroscopic" stage of coding, and writing instructions as the "static or microscopic" stage.

h H. Goldstine to H. Curry, Oct. 3, 1945, in the collection "ENIAC Patent Trial Collection" in the University of Pennsylvania archives.

1947 the noun "program" was becoming firmly established to refer to coded sequences of instructions.[i]

In the 1950s, "coding" acquired a more specific meaning as the most mechanical part of programming—tasks such as looking up numerical codes corresponding to particular instructions. "Coder" endured in some organizations as a job title for the most junior programmers. In the last decade or so it has been revived as an expression of geek pride, perhaps as a reaction against the trend toward increasingly abstract job titles for software developers such as "software engineer" or "solutions architect."

New Meanings

Space does not permit us to follow the further evolution of the concepts of program and programming in any detail, so instead we will flag a few key aspects of the subsequent history. The first is the distinction, sometimes made in the late 1940s and 1950s, between a "stored program" loaded into internal memory and an "external" program wired onto plug boards or read one instruction at a time from tape.[5]

In the early 1950s, "automatic programming" systems such as assemblers complicated the concept of program. The program actually executed by a computer, a string of numerical codes, became something that could be automatically generated from a different kind of input, commonly known as "pseudocode." This introduced two levels at which a program could be viewed, and the relationship between the levels was widely understood as one of translation.[8]

As the automatic programming systems became more complex, linguistic metaphors continued to gain currency. The FORTRAN system, released by IBM in 1957, translated mathematical expressions, data structure definitions, and control structures into executable programs. FORTRAN is remembered as the first widely used "high-level programming language." Donald Knuth and Luis Trabb Prado explored the many obscure and experimental systems that led up to this milestone, concluding that Konrad Zuse's Plankalkül, a proposal for which was published in 1948, was the first public description of the concept of a programming language.[7]

The increasing need through the 1950s to run programs on machines of different types led to a search for a "universal" programming language, culminating in the publication of the Algol proposals in 1958–1960. An Algol program had no association with a particular computer and, after *Communications* standardized on the language for its "Algorithms" department, was often intended primarily to be read by humans rather than executed by machines. Nowadays usage has widened to the point where the word "program" can refer equally to the "source code" written in a high-level language and the "object code" into which it is translated for execution on a particular machine.

Conclusion

Before the 1940s nobody talked about programming computers and no computers had what we consider to have been the original and fundamental meaning of programmability: the ability to automatically execute a specified series of operations. While this sense of programming could be applied to machines able to execute a series of coded arithmetic operations but not able to automate complex control structures, the fact is the earliest references to "programming" appear in the context of the first computer able to automatically execute nested loops and conditional branches: ENIAC.

We see ENIAC's control innovations as pragmatic engineering responses to the need for mechanisms that, unlike paper tape or human intervention, could keep up with its unprecedented electronic speed of operation. Its designers relied on problem-specific wiring to route networks of "programming pulses" around the machine. In the "First Draft" design for EDVAC, von Neumann extended the coding approach of the relay computers, designing a single instruction set that could express not only sequences of arithmetic operations but also the control structures pioneered on ENIAC. The EDVAC code unified arithmetic and control, programming a single set of logical circuits. It is in this context that people began calling the coded instructions a "program," a usage that evolved from related but distinct meanings of "program" and "programming" within the ENIAC project.

This provides a rather different view of the invention of computer programming, and its relationship to logic, from the widely held assumption that computer development in the 1940s was guided directly by the theoretical work of Alan Turing. In that view of history, a metaphysical attraction to the idea of "universality" inspired a competition amongst computer builders to be the first to check a box labeled "Turing complete." Von Neumann's design for EDVAC was elegant and its generalization and simplification of ENIAC's control capabilities unquestionably reflected his grounding in mathematical logic. The usefulness of a computer able to tackle many different kinds of calculations was certainly appreciated by the creators of the first automatic computing machines. The computer builders of the 1940s and 1950s adopted EDVAC's new design paradigms because they provided an efficient way to automate real machines, running real computations to solve real problems. C

References

1. Bromley, A.G. Babbage's analytical engine plans 28 and 28a—The programmer's interface. *IEEE Annals of the History of Computing 22*, 4 (2000), 5–19.
2. Eckert, J.P., Mauchly, J.W., Goldstine, H.H., and Brainerd, J.G. Description of the ENIAC and Comments on Electronic Digital Machines. AMP Report 171.2R. Distributed by the Applied Mathematics Panel, National Defense Research Committee, (Nov. 30, 1945). Moore School of Electrical Engineering, Philadelphia, PA, 1945.
3. Haigh, T., Priestley, M., and Rope, C. *ENIAC in Action: Making and Remaking the Modern Computer.* The MIT Press, Cambridge, MA, 2016.
4. Haigh, T., Priestley, M., and Rope, C. Los Alamos bets on ENIAC: Nuclear Monte Carlo simulations, 1947-1948. *IEEE Annals of the History of Computing 36*, 3 (July–Sept. 2014), 42–63.
5. Haigh, T., Priestley, M., and Rope, C. Reconsidering the stored program concept. *IEEE Annals of the History of Computing 36*, 1 (Jan.–Mar. 2014), 4–17.
6. Knuth, D.E. Von Neumann's first computer program. *ACM Computing Surveys 2*, 4 (Dec. 1970), 247–260.
7. Knuth, D. and Prado, L.T. The early development of programming languages. In *A History of Computing in the Twentieth Century*, N. Metropolis, J. Howlett, and G.-C. Rota, Eds., Academic Press, New York, 1980, 197–273.
8. Nofre, D., Priestley, M., and Alberts, G. When technology became language. *Technology and Culture 55*, 1 (Jan. 2014), 40–75.

Thomas Haigh (thaigh@computer.org) is an associate professor of information studies at the University of Wisconsin, Milwaukee, and chair of the SIGCIS group for historians of computing.

Mark Priestley (m.priestley@gmail.com) is an independent researcher into the history and philosophy of computing.

i For example, in S. Lubkin, "Proposed Programming for the EDVAC" (January 1947), in box eight of the collection "Moore School of Electrical Engineering Office of the Director Records, 1931–1948" in the University of Pennsylvania archives.