Roots of program revisited

Liesbeth de Mol, Maarten Bullynck

To cite this version:

Liesbeth de Mol, Maarten Bullynck. Roots of program revisited. Communications of the ACM, 2021. hal-03287324

HAL Id: hal-03287324
https://hal.science/hal-03287324

Submitted on 22 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Roots of "program" revisited[1]

Liesbeth De Mol[2] and Maarten Bullynck[3]

Today, it is a widely accepted thesis amongst historians and computer scientists that the *modern* notion of computer programs has its roots in the work of John von Neumann. This is symptomatic of a general tendency amongst academic computer scientists to search for the foundations of their field in logic and mathematics and, accordingly, also its historical roots. This is a distorted view of what happened: at best, the modern computer was driven by concerns of applied mathematics and developed by a collective of people (mathematicians, engineers, physicists, (human) computers, etc.). We will not repeat why, in computing, history is reshaped in function of disciplinary identity [2,13]. Instead, we will revisit the origins of the word "program" and argue for the need of a deeper historical understanding, not just for the sake of academic history but for the sake of the field itself.

The notion of "program" is a fundamental one. In the flux of historical time and space, "program" underwent significant changes and has different connotations today when compared to the 1950s. Indeed, today, other words are often used instead: "software", "apps" or "algorithms" (as in "ethics of algorithms"). Moreover, "program" means different things to different people: a logically-minded computer scientist will have a different understanding than a software engineer. Nonetheless, as soon as one starts to speak about the historical origins of the term, this plurality of meanings disappears to be replaced by only one, viz. the "stored program". This is anchored in another historical narrative: the modern computer originates in the "stored-program" computer. While this latter notion has been historically scrutinized [6], the origins of "program" have not been looked at independently of that notion. So what is the classical story here?

A narrative

In the mid 1940s a group of engineers of the *Moore School of Electrical Engineering,* led by *John Mauchly* and *Presper J. Eckert,* designed and constructed ENIAC, a large-scale and high-speed machine that would become *one of the* first computers. Originally, it was a parallel and electronic machine with loops and conditionals, and could, essentially, compute any problem provided that its memory would have been unlimited. However, unlike some other large-scale calculators of the time, like the relay-based ASCC/Harvard Mark I or the Bell Lab machines, problems were not set-up via coded instructions on punched cards or tape but were directly wired on the machine. In a sense, one had to reconfigure the whole machine every time it had to compute another problem. By consequence, setting-up a problem was a time-consuming and error-prone process. In order to deal with such efficiency issues, ENIAC was converted to emulate a "stored-program" machine. However, unlike EDSAC for instance, instructions could not be modified since programs were executed from preset switches (or, alternatively, punched cards)

It is here that von Neumann enters the story. A few months after he got involved with ENIAC, in the spring of 1945, he wrote the famous "The first draft of a report on the EDVAC" which is considered the blueprint of the modern computer. It is then often assumed that the first "modern" programs must be those that ran on EDVAC-like machines, that is, machines like the converted ENIAC [4,6,7]. This goes hand-in-hand with the idea that the roots of our modern conception of program should be sought with von Neumann.

---

1 This is a preprint of a paper published in the Communications of the ACM, available here: https://cacm.acm.org/magazines/2021/4/251342-roots-of-program-revisited/fulltext

2 CNRS, UMR 8163, Université de Lille. This research was supported by the ANR PROGRAMme project ANR-17-CE38-0003-01

3 Université de Paris 8, UMR 8533 IDHE.S.

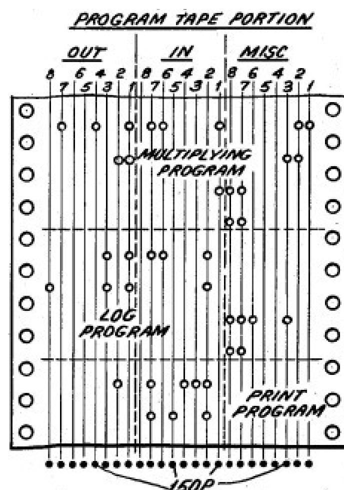
But of course, just as “code” was already in use before it was introduced in the computing context (e.g. Morse code), also “program” (or in Bristish spelling “programme”)[4] was an already existing word. No new term was invented at the time. Instead it was used as a generic term to refer to the planning and scheduling of events: an advance notice, an itinerary, something that is written *before* some activity happens, orders it and, so, pre-(in)scribes (προ-γραμμα) it. Typical examples are: a theater program, a training schedule, a research program or a production plan [5,6]. This notion was then picked up in the context of radio broadcasting to refer to radio programs. Presupposedly, Mauchly transposed this common term to the more specific engineering discourse of ENIAC and it was only with the introduction of “stored-programs” that the term really gained its current meaning [4,6,7]. Von Neumann himself, however, never really used it – he preferred the more common terminology of preparing, planning, setting-up and coding a problem.

Deconstructing the narrative

We have found that “program” was already part of an extensive engineering discourse, going beyond that mentioned in the existing literature. First of all, in the context of radio engineering, the growing complexity of the broadcasting network increased the need for automation, especially when it concerned the scheduling of radio programs in different networks for different stations at different times and which had to be handled at so-called “switching points”. This resulted in a discourse in which “program” steadily transposed from radio programs to the technology *itself* and so one sees the emergence of terms like “program circuits”, “program trunks”, "program switching", "program line", “program loop”, “program transfer”, etc.[5] These are exactly the kind of terms appearing in the ENIAC context.

More importantly, we have found that there is yet another engineering discourse originating in so-called “programme clocks” or “program clocks”, a device first developed in the 19th century and used to “*furnish a convenient and practical clock, that may be set to strike according to any required programme”*.[6] Program(me) clocks were devices that could be set according to a given schedule or program to ring at preset times. This was very handy, for instance, for a factory work floor, railway stations or a school. In other words, they *automated* time schedules and work programs. From the first clocks onwards, one sees the steady development of a more general technology of “program devices” or "program machines" used in a variety of applications: a paper cutting machine, a washing machine, a calculator, etc.[7] Here "program" comes to stand for the automatic carrying out of a sequence of operations or as an automated scheduler.

This technology came to be used also for calculating machines in the late 1930s and early 1940s and so also appeared in that discourse, for instance, in the context of the IBM ASCC/Harvard Mark I machine. This electromechanical large-scale calculator is mostly associated with Howard Aiken, a Harvard physicist, but was designed and built by IBM engineers.



4 Note that in American spelling, “pro[...] ms and, say, radio programs. In the British spelling, an explicit distinction was ma[...] omputer programs, and “programme” as used in the context of radio or theater [...]
5 See for instance US patents number [...] 70.
6 See U.S. patent nr. 98678
7 See for instance US patents numbers [...]

Fig. 1: Graphical representation of the “program tape” from the ASCC/Mark I patent, Lake et al 1945, US patent number 2,616,626

The operations of that machine were controlled by the “control tape” where the sequences of operations were coded with punched holes. But while “control tape” was the standard term used once the machine was put into operation at Harvard, the original IBM patents show traces of another terminology where the control tape was also called a “program tape” and where the sequences of operations, at some points, were called “programs” instead of “sequences” (See Fig. 1). This terminology is due to the IBM engineers involved with the design of the machine, notably, James W. Bryce and Claire D. Lake. In fact, Bryce already had a patent in which a “program device” is introduced (US patent number 2,244,241) that was capable of automatic transfer of control and other operations. Also for the ENIAC, Mauchly’s original short proposal for an “electronic computer” refers to a “program device” [10]. It is from there that the term in ENIAC developed.

Some have claimed that earlier uses of “program” in relation to ENIAC were much more restricted and referred *only* to specific programming circuitry in a (control) unit [5]. This does not take into account this more general discourse which, by that time, had become common among engineers working on *automatic* control, both within and outside of the context of large-scale calculators. Combining that with the radio broadcasting discourse where “program” terminology had been transposed to the technology itself, explains why, in ENIAC, “program” had different semantic extensions and referred both to individual (control) units (as in “program switches”); smaller pieces of an entire program (as in “program sequences”); or the complete schedule that organizes program sequences (as in a *“complete program [for which] it is necessary to put [the] elements together and to assign equipment in detail”* [1]*)*. ``Program’’ then refers to how automatic control, locally or globally, is organized. The semantics of the ``program device’’ discourse is still at play here, but generalizes from the sequencing of operations to include also the scheduling of sequences of operations.

This is still evident in Hartree’s later definition (1949) of “programming” where this notion is used with reference to any “large automatic digital machine”: *“programming is the process of drawing up a schedule of the sequence of individual operations required to carry out the calculation”* [8, pp. 111-112]. The main difference between programming ENIAC before and after its conversion to an EDVAC-like machine, is that in the second case the set-up is automated through “*a 100-way switch*” [8, p. 87] where each position of the switch corresponded to a different “computing sequence”.[8]

8 Of course, subsequent programming practices would impact later definitions. So, for instance, in the EDSAC, an EDVAC-like and so serial machine with a symbolic assembly system, the definition of programs shortened to: “*A sequence of orders for performing some particular calculation”* [14]. In other contexts, where flowcharting played an important role, emphasis was more on the planning aspects of programming, partially referencing back to earlier practices of human computation (see the 1954 ACM Glossary by Grace Hopper).

To put it differently, the *general* understanding of “program” was first grafted onto the existing discourse on program devices *not* on specific techniques for implementing them. The addition of using coded instructions stored on cards, tape or flipflops rather than the technique of wiring a program through plugboards was, from that perspective, non-essential for the meaning of “program”. This, in a sense, should not be surprising: while “programs” are very much determined and dependent on the computational technology on which they are ultimately implemented and ran, that need not mean that understandings of “program” should be reduced to properties of and possibilities offered by those technologies. If we would have done that, we would have never had, say, concurrent programs, virtual machines or Docker containers.

Why this matters

The 2012 Turing centenary made clear that the academic computing field tends to construct a storyline where the presumed theoretical foundations of the field coincide with its historical foundations (the “first” computers and the “first” programs). This strengthens a computing discipline where one often cares more about formalism than about actual programming [11] and contributes to a growing “communication gap” between different communities. This affects not just research and education policies but also how we understand this field we call computing [3].

As we showed, “program” did not coincide with the “stored-program” concept, rather it naturally evolved from an engineering context. Program devices for automatic control of operations were developed first for scheduling activities or communications, but were then applied to computing machines as well. In this context, a transfer of meaning happened, preparing the ground for our modern notion***s*** of program. Should we derive from this that, actually, computing should be understood first of all as an engineering discipline? No. The point is that as soon as one confines oneself to the perspectives offered by one discipline only, one misses out on the richness of the field as a whole and so lacks a basic understanding: computing is *not* mathematics, it is *not* engineering, it is *not* logic, it is *not* science but a field on its own and one which should, perhaps, not be reduced to the confines of disciplinary thinking (which is, itself, a construction of the 19th century). Abiding by such confinement may lead to errors and failed opportunities, as Hennessy and Patterson recently pointed out [9] with respect to software design and hardware architecture.

“[W]hen experience is not retained [...] *infancy is perpetual. Those who cannot remember the past are condemned to repeat it”* [12] History can and has been used to reinforce confines but it can also be used against them. We must not see our historical legacy as a burden, but as the natural environment to think about the future.

References

1. Curry, H.B. and Wyatt, W., *A study of inverse interpolation on the Eniac*, Aberdeen Proving Ground, Maryland, Report nr. 615, 19 August 1946.

2. Bullynck, M., Daylight, E.G., and De Mol, L., “Why did computer science make a hero out of Turing?”, *Communications of the ACM*, vol. 58, nr. 3, 2015, pp. 37–39.

3. Denning, P.J. and Tedre, M., *Computational thinking*, MIT Press, 2019.

4. Grier, D. A., `The ENIAC, the Verb “to program” and the emergence of digital computers', *IEEE Annals for the history of computing*, vol. 18, nr. 1, 1996, pp. 51–55.

5. Grier, D.A., `Programming and planning', *IEEE Annals for the history of computing*, vol. 33, nr. 1, 2011, pp. 85–87.

6. Haigh, T., Priestley, M., and Rope, C. *ENIAC in action. Making and remaking the modern computer*, MIT Press, 2016.

7. Haigh, T. and Priestley, M., `Where code comes from: Automatic Control from Babbage to Algol', *Communications of the ACM* , vol. 59, nr. 1, 2016, pp. 39–44.

8. Hartree, D.R., *Calculating instruments and Machines* , Urbana, University of Illinois Press, 1949.

9. Hennessy, J.L. and Patterson, D.A., `A new golden age for computer architecture,' *Communications of the ACM*, vol. 62, no. 2, pp. 48--60, Feb. 2019.

10. Mauchly, John, `The use of high speed vacuum tube devices for calculating,' Moore School of electrical engineering, University of Pennsylvania, August 1942.

11. Noble, J. and Biddle, R., "Notes on postmodern programming," *ACM SIGPLAN Notices*, vol. 39, nr. 12, 2004, pp. 40-56.

12. Santayana, G., *Reason in Common Sense*, volume I of *The life of Reason,* New York, Charles Scribner's Sons, 1905.

13. Tedre, M., *The science of computing:Shaping a Discipline*, CRC Press, 2014

14. Wilkes, M.V. and Wheeler, D.J., Gill, S., T*he preparation of programs for an electronic digital computer*, second edition, Addison-Wesley, 1957.

These different uses of “program” circulated around ENIAC and there are different reports in which one can find this terminology. Even though these reports were often written *after* the *EDVAC report* they also referred to “programs” for a non-EDVAC-like machine. The addition of using coded instructions stored on cards, tape or flipflops rather than the technique of wiring a program through plugboards is, from that perspective, non-essential for the meaning of program. It matters *only* from the efficiency perspective sketched before. To put it differently, the *general* understanding of program was grafted onto the existing discourse on program devices *not* on specific techniques for implementing them.

As is clear from these cases, within the early ENIAC, the semantics of ``program" and ``programming" is not fixed but shifts and moves between the local control units to the overall program that is organized as a schedule or complex hierarchy of program sequences. In other words, the semantics of the old discourse around ``program devices" and problems of sequencing \textit{and} scheduling is still at play here but rather than being about the sequencing of operations (as was the case with the ASCC/Mark I) it shifted to the sequencing and scheduling of sequences.

The terminology had not made full abstraction yet from the physical machine. This is explained by the fact that ``program" did not refer to a code (which was absent in early ENIAC) but to the organization of the machine itself. This could only change with the converted ENIAC and the EDVAC design that underlined it and in which the set-up process was automated by the introduction of the so-called ``order code". ``Setting-up" a problem steadily shifted to the ``coding" of a problem. ``Programming" with its reference to the organization of program sequences, shifted again to refer much more to the practice of developing efficient methods that ease the burden of organizing a computation into ``stages", ``sequences" or, what became the more popular term, subroutines. The different strategies that were developed to attack this problem resulted in a number of different concepts and conceptions of programming and coding.

Also later this meaning of program persists. Douglas R. Hartree, in his popularizing book on computing machines [7], defines “program” as: “*a sequence of operation[s] for a particular calculation*” (Hartree 1949). Later, Wilkes, Wheeler and Gill [11] define “program” as: “*A sequence of orders for performing some particular calculation”*. Here, programs are intended for the EDSAC, which was an EDVAC-like machine. However, “program” refers to program-controlled computing, which has to be distinguished from stored-program control. The former is automatic control over a computation through a program, whatever its materiality, the latter is a program stored in the computer's memory, controlling the operations of the machine in the same medium.

Another understanding of “program” which starts to appear in the late 1940s is a definition of program as a “plan”. Indeed, in the ACM Glossary of 1951, which was compiled by Grace Hopper, programs are defined as: “*a plan for the solution of a problem*”. This was picked up in several glossaries. Note that such definitions are historically connected to the earlier practices of human computation where one spoke of a computation plan.

Thus “program” as used in the discourse *around* ENIAC and afterwards was not anchored in the EDVAC report but in a more general understanding of “program” as a sequencing and scheduling of operations. The EDVAC report then described a technology for achieving this more efficiently.

This, in a sense, should not be surprising: while “programs” are very much determined and dependent on the computational technology on which they are ultimately implemented and ran, that need not mean that understandings of “program” should be reduced to properties of and possibilities offered by those technologies. If we would have done that, we would have never had, say, concurrent programs, virtual machines or Docker containers.