Revisiting the roots of “programs” and why you should care.

Liesbeth de Mol, Maarten Bullynck

Revisiting the roots of “programs” and why you should care.

Liesbeth De Mol
Maarten Bullynck

Please do not circulate. This is a draft.

Today, it is still 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 for a general tendency amongst academic computer scientists to view the foundations of their field as logico-mathematical and, by consequence, also its historical roots. This, despite clear evidence that, at best, the modern computer was driven by concerns of applied mathematics and developed by a collective of people, mathematicians but also engineers, physicists, (human) computers, etc. Hence, claiming that the most important developments in computer science have their origins, first of all, in the work of mathematicians and logicians, is a distorted view on history. It is not our aim here to repeat some of the arguments why, in computing, one is inclined to reshape history in function of disciplinary identity (Tedre 2014, Daylight et al 2015). Instead, we would like to revisit the origins of “program” and argue for the need for 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 computing. In the flux of historical time and space, “program” underwent significant changes and so “program” in the 1950s would have different connotations from “program” today. Indeed, today one often even uses 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 of “program” 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 to some extent,[1] 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* designed and constructed the ENIAC, a large-scale and high-speed machine that would become *one of the* first computers. Originally, it was a highly 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 at the time, like the relay-based ASCC/Harvard Mark I or the Bell Lab machines, problems were not set-up on the machine via code 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. Moreover, the “size” of a problem that could be computed was, in a certain sense, determined by the “size” of the machine. In order to deal with these issues, ENIAC was soon converted in a kind of “stored-program” machine. It is here that John 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 (Haigh and Priestley 2016). This goes hand-in-hand with the idea that the roots of our modern notion of program should be sought with von Neumann.

1 One interesting outcome of that work (Haigh et al 2014) is the observation that the term “stored-program” was not even used until the late 1940s and that it has received different meanings in different times. For those reasons we will gloss the tern “stored-program” here.

But of course, just as “code” was already in use before it was introduced in the computing context, also “program” was an already existing word. No new term was invented at the time. In fact, as is quite well-known, the word “program” became our modern word because it was extensively used around ENIAC *before* and *after* von Neumann entered the scene. Von Neumann himself, however, never really used it – he preferred the common terminology of preparing, planning, setting-up and coding a problem for the EDVAC.[2]
Before ENIAC it is assumed that the term “program” was not used in a computational setting. Its earlier meanings, according to the entries of the *Oxford English Dictionary (OED)*, referred to such things as: an advance notice, an itinerary, something that is written *before* some activity happens, orders it and, so, pre-scribes it. A typical example is a theater program or a training schedule. This notion was then picked up in the context of radio broadcasting to refer to radio programs.[3]
In ENIAC, the notion was introduced by John Mauchly, together with Presper J. Eckert, one of the main ENIAC engineers. It is then assumed that he transposed this common term to a more specific engineering discourse of the ENIAC.

Deconstructing the narrative

We have found clear evidence that, even before ENIAC, there was already an extensive engineering discourse around “program” that goes beyond that mentioned in the OED. First of all, in the context of radio engineering, the growing complexity of the radio broadcasting network increased the need for automation, especially when it concerned the scheduling of radio programs in different networks for different stations 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 switches", ``program loop", ``program transfer", etc. It are exactly these kind of terms one also finds in the ENIAC context from the start.

More importantly, we have found that there is another engineering discourse using “program” terminology and has its roots in so-called ``programme 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”* (Estell 1870). Thus, ``program clocks" were devices that could be set according to a given schedule or program so that it can ring at preset times. This was very handy, for instance, for a factory work floor, railway stations or a school. From the first clocks onwards one sees the steady development of a more general technology of so-called “program devices” or ``program machines" to be used in a variety of applications such as a paper cutting machine, a washing machine, an automatic telephone and responding machine or a calculating machine. In that context, ``program" comes to stand for the automatic carrying out of a sequence of operations or as an automated scheduler.

These different meanings of “program” circulated amongst engineers. Thus, it is highly plausible that Mauchly relied on this existing discourse on the automatic sequencing and scheduling of operations when he introduced the term around ENIAC. This is further substantiated that, as we found, he was not the only one to do so. In an earlier design stage, “program” was also used for the IBM ASCC/Harvard Mark I machine, an important electromechanical large-scale digital calculator. The machine is mostly associated with the name of Howard Aiken, a Harvard physicist, but was designed and built by IBM engineers. The operations of that machine were controlled by the “control tape” in which the sequences of operations were coded with punched wholes. 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

2 “Coding” and “code” were extensively used in the context of the Harvard Mark I which von Neumann knew very well; the idea of “setting-up” a problem on a calculating machine was used in the context of the differential analyzer which he also knew very well.

3 In (Grier 1997) it is claimed that “program” before ENIAC also referred to “any electrical signal” but we could not find evidence for this in the sources provided.

"program tape" and where the sequences of operations, at some points, were called "programs" (See Fig. 1) instead of "sequences".[4]

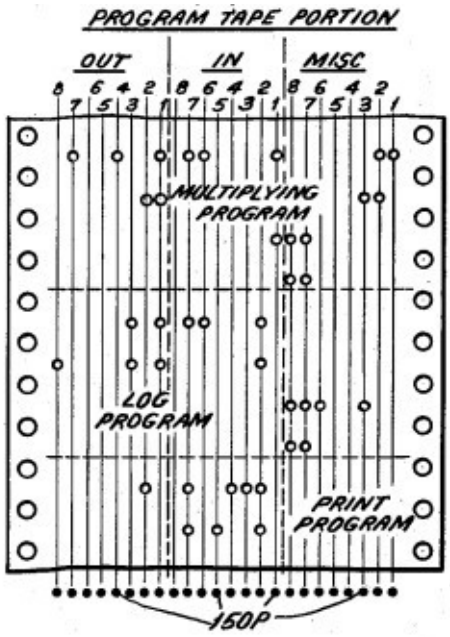

Fig. 1: Graphical representation of the program tape from the ASCC/Mark I patent (Lake et al 1945)

This terminology is due to the IBM engineers involved with the design of the machine, amongst others, James W. Bryce and Claire D. Lake. In fact, Bryce already had a patent, predating his first meetings with Howard Aiken, and in which one main technology is a "program device" which is used, amongst others, to do an automatic transfer of control.[5]

Some have claimed that earlier uses of "program" around ENIAC, before von Neumann became involved, were much more restricted and referred only to "a single operation set up on one of its units" or "the [programming of] operations of the internal circuitry of that unit" (Haigh and Priestley 2016) This does not take into account this more general discourse which, by that time, had clearly become widespread among engineers working on automatic control. This is clear in the ASCC/Mark I example we discovered but also, for instance, in an Army report written by Eckert, Mauchly and Brainerd to describe the main ideas of the ENIAC. They state that the "*purpose […] is to indicate one of many possible ways of setting up a program for the step-by-step solution of a pair of equations commonly used in external ballistics*". A device called the "master programmer" was used for the sequencing and scheduling of loops and conditionals[6] and was an integral part of the original design of the ENIAC. This goes to show how "program", from the start, referred not just to individual (control) units (as in "program cables") or smaller pieces of an entire program (as in "dummy programs"), but also to the complete problem as a "set-up" on the machine.[7]

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 referred to programs for non-EDVAC-like machine but which do have loops, subroutines (closed and open), conditionals, sequences of operations, etc.[8] The addition of using

4 There were actually two patents, one by the IBM engineers, and one for the Harvard machine.

5 Note that Bryce was, however, not one of the inventors listed on the ASCC patent. He was, however, very much involved with its design and had the first contacts with Aiken.

6 Remember that in its original constellation ENIAC was a parallel machine and so scheduling was one basic problem to be counted with.

7 Note that also the term "set-up" was frequently used instead of "program".

8 Apart from the original (parallel and decentralized) ENIAC, also the Bell model V machine or IBM calculators were considered.

code rather than the technique of wiring a program is, from that conceptual perspective, non-essential. It matters *only* from the efficiency perspective sketched before. Or, to put it differently, the *general* understanding of "program" was much more related to the existing discourse on program devices than to specific techniques for implementing them.

Also later this meaning of program persists. Douglas R. Hartree, who was involved with ENIAC and several other computing devices, gives an explicit definition of "program" in his popularizing book on computing machines: "*a sequence of operation[s] for a particular calculation*" (Hartree 1949). Later, Wilkes, Wheeler and Gill in their influential *Programs for an electronic digital computer* define "program" as: "*A sequence of orders for performing some particular calculation"*. Of course, in this latter context, programs *are* intended for EDSAC and thus for an EDVAC-like machine but even then, it is clear that this is not part of the definition of program per se. They point to the aspect of program-controlled computing, which has to be distinguished from stored-program control. The former is automatic control 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*". A similar understanding can be found in another paper by Maurice Wilkes titled *Programme Design for a High-Speed Automatic Calculating Machine.* It was also 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 so much 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. Also in later years, "program", both on EDVAC-like machines and non-EDVAC-like machines, was defined in a more general manner, often with reference to the older discourse.

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

Since the 2012 Turing centenary it has become clearer that the academic computing field has a tendency to construct for itself a storyline whereby the presumed theoretical foundations of the field coincide with its historical foundations (the "first" computers and the "first" programs). This is not without effects but only strengthens an academic computing field where one often cares more about formalism than about actual programming (methods) (Noble and Biddle 2004) and thus contributes to a growing "communication gap" between different communities. Moreover, if one assumes that one approach prevails in the computing field (be it logic or something else), this also has its effects, not just on research and education policies but also on how we understand this field we call computing.

As we showed in this viewpoint, the notion of "program" did not originate with von Neumann or with the stored-program concept, rather it naturally evolved from an engineering context. Program devices for automatic control of operations were developing first for scheduling activities or communications, but were then applied to computing machines as well. In this context, the transfer of meaning happened, preparing the ground for our modern notions 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 of computing 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).
History can and has been used to reinforce such confines but it can also be used against them. We thus would like to make an appeal to you, reader, to not just dismiss this piece as just another nice historical anecdote, but to pick it up as an opportunity to question your own disciplinary confines and assumptions. We hope that by doing so you will not only render your own history more transparent but, perhaps, also make an effort to bridge some communication gaps.

References

Daylight, E.G., Bullynck, M. 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.

Estell, S.F., `Improvements in programme-clocks,' US patent nr 98678, patented January 11, 1870.

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.

Haigh, T., Priestley, M., and Rope, C. `Reconsidering the stored program concept.' *IEEE Annals of the History of Computing*, vol. 36, nr. 1, 2014, 4–17.

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.

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

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

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