The Myth of the Coder

Maarten Bullynck, Liesbeth de Mol

The Myth of the Coder

Maarten Bullynck and Liesbeth De Mol

It is a stock element of narratives of the early computing days (1950s) to distinguish between programmers and coders, the latter considered a "lowly technician" routine job of converting flowcharts or pseudo-intructions into coded machine instructions (see, e.g., [4, p. 168; 7]). This division of labour between coders and programmers is often overlaid with social distinctions of education, gender and race [1]. Researching the early uses of the words "programming" and "coding" [5], however, we were confronted with the near absence of evidence for this coder/programmer distinction. While the activities of coding and programming can be more or less clearly distinguished, the activities do not map onto different jobs, but rather are performed by the same person(s). It thus seemed that the distinction between the coder and the programmer pertains not to the reality but to the mythology of early computing. It appeared in the wake of automatic programming in the 1950s, creating a discourse that is influential until today.

1. *The origin of the myth of the coder*

The distinction between coder and programmer originated in the famous report on *Planning and Coding of Problems for an Electronic Computing Instrument* written by Herman H. Goldstine and John von Neumann. They give four hierarchical stages for planning and coding. At the top is the "mathematical stage" for the derivation of the algorithmic form of a mathematical problem. This stage was exclusively for the mathematician and "has nothing to do with computing or with machines" [8, p.19]. After this "the coding proper can begin", subdivided into three stages:

> 1) a *macroscopic* or *dynamical* stage capturing the dynamic aspects or the flow of a computation including loops, conditionals and address modifications formalized by flowdiagram
> 2) a *microscopic* or *static* stage which concerns the actual coding of every single operation box of the flow diagram
> 3) the last stage which consists (mostly) of assigning memory locations and conversion to binary

Drawing up a flow diagram (stage 1) should be "a routine matter" for "every moderately mathematically trained person" [8, p.20]. As for the static coding (stage 2), "a moderate amount of experience" will make it "a perfectly routine operation". After some preparation coding becomes "filling-in operations that can be effected by a single linear passage over the entire coded sequence" [8, p.21]. The distinction between stage 1 and 2 would provide the blueprint for the distinction between programming (often equalled to planning or flowcharting) and coding (that is, translating a flowchart into coded instructions for the machine).

Why did von Neumann and Goldstine assume the planning and coding could be easily separated? This is explained by their experience with the organization of labor in manual computation (cf. [9]). Goldstine knew the organization of the (manual) computation of firing tables at Aberdeen Proving Ground. These had been profoundly reshaped shortly after World War I and a clear-cut division of labor had been installed [3]. There was the mathematical theory and its algorithms, then there were "computation sheets" (Form 5041) that laid out step-by-step what the human computers (soldiers or locals with reasonable good grades in arithmetic) had to do. Following the control orders of the computation sheets, the human computers had to look up values in tables, look up logarithms and perform additions and subtractions, the intermediate results were written on a "data sheet", the end results were copied down on a "trajectory sheet" (Form 5041). For the people at the Ballistic Research Laboratory, this organization of manual calculation translated with ease to electromechanic and electronic machines [11, p. 9 and 7]. Even on the ENIAC rewired to simulate a stored-program computer (1948), this division between programming and coding had been tried

with some succes, as Klari von Neumann's coding of the Monte Carlo flowcharts shows [10, Chapters 8 & 9].

2. ***The coder absconded***

The *activities* of programming and coding could be distinguished easily, the former was planning on paper using flowcharting[1] or some pseudocode, whereas the latter was the translation into coded instructions on some medium to control the machine. Though they could be discerned easily, they could not readily be separated from one another. This was the experience they would make at UNIVAC, the company Eckert and Mauchly started after having built the ENIAC for the military [16].

Before delivering their first machine, they had stipulated a distinction between the jobs of the coder and the programmer (1949) [7, p. 39 & 251]. A year later, "standard flow chart and coding symbols and practices" were introduced to facilitate review by "a person other than the person who originally did the charting and the coding" [14, p.1]. Although UNIVAC had a detailed flowcharting manual following von Neumann and Goldstine's formalism, in practice flow diagrams were used as "a potent means of communications", filling the flow diagram's boxes with anything, from "mathematical symbols to sentences" [2, p. 17-1]. Once the UNIVAC 1 computer was ready (1951), any distinction between coder and programmer was brushed off the table. When Ohlinger from Northrop Aircraft Company asked J.L. McPherson from UNIVAC: "how many programmers and coders were employed in order to keep UNIVAC busy full time?" McPherson replied: "We do not distinguish between programmers and coders. We have operators and programmers." [6]

This is consistent with what we found in other places in the 1950s, both at universities and in industry, where "programmers" did both the planning and the coding part of programming. When the question was raised during MIT's 1954 Summer Conference what groups distinguished between coder and programmer, only 8 out of over 50 participants said yes [2, p. 17-1], only one company adhered to this strict division of labor, Douglas Aircraft Company.

This observation is confirmed by reports commissioned by the U.S. government from 1957 onwards to assess the impact of automation and to map the needs of the government in matters of "automatic data processing" (ADP). To establish grade level distinctions and according pay rates, the Department of Labor did a broad survey in 1959 and presented a list of 13 "occupations in electronic data-processing systems". While a division of labor is implied by the functional organization chart of these occupations (fig. 1), the job of the coder is notable by its very absence. Instead, a distinction is made between a systems analyst, a programmer and a coding clerk. The first defines the problem and its requirements, the programmer then "designs detailed programs, flow charts, and diagrams indicating mathematical computations and sequence of machine operations necessary to copy and process data and print solution" [17, p. 19], though a chief programmer may be there "to assign, outline and coordinate" the work of the programmers [17, p. 20-21]. The coding clerk "convert[s] items of information from reports to codes for processing by automatic machines using a predetermined coding system" [17, p. 10]. A report by Weber and Associates on salaries in ADP from 1960 [18] equally distinguishes between Lead Programmer, Programmer Senior, Programmer A, B and C in the programming departement, all involved in both flowcharting and translating of charts into coded instructions. Again, the separate job of "coder" that would convert flowcharts into coded machine instructions is absent.[2]

---

1 The definitions vary a lot: Whereas the ACM 1954 glossary states that programming " consists of planning and coding", the IBM glossaries are generally more restrictive, e.g., " Programming and flow charting are synonymous – the remainder is mere coding".

2 A"coder" does appear in the Auxiliary Equipment department [18, p.22] , but this is actually a "coding clerk" and defined in exactly the same terms as [17, p. 10].

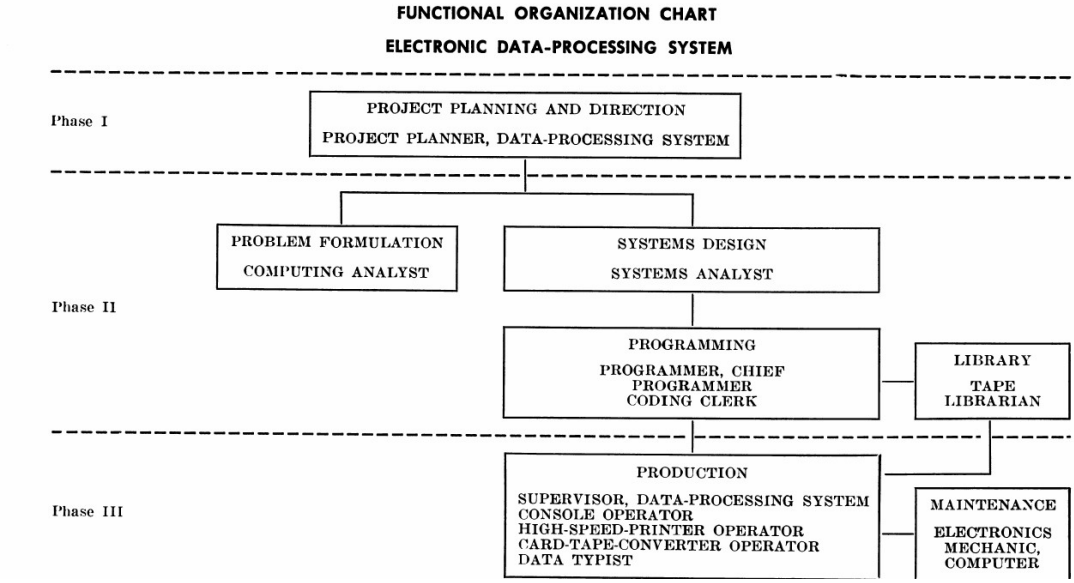

Figure 1: Functional organization chart of ADP occupations [17]

3. How did the coder enter computing mythology?

How did the idea of the coder (vs. the programmer) than take root and why? If one looks at trade magazines such as *Computers and Automation* or *Datamation* in the 1950s, a (human) coder is rarely mentioned (on average 3-4 times a year). But in 1955, there are no less than 28 occurences of "coder" in just three articles, accounting for 60% of all occurrences between 1954 and 1960. All three articles were authored by Grace Murray Hopper and her team. In charge of the automatic programming department of UNIVAC since 1954, Hopper became a relentless advocate for the automatization of programming, and very influential through her public speeches in the 1950s.

In May 1954, Hopper introduced the automation of the coder as an essential point. She described how "ten years ago, a programmer was, of necessity many things." [12, p.1] But with the "increase in the number and speed of computers" came "specialization". This specialization is borne out in the birth of "specialists", notably "analyst, programmer, coder, operator and maintenance man", separated and communicating only via special types of aids such as flow diagrams. Hopper admitted that, actually, the "distinction between a programmer and a coder has never been clearly made." Indeed:

> Coder was probably first used as an intermediate point of the salary scale between trainee and programmer. A "programmer prepares a plan for the solution of a problem: […] One of his final results, to be passed on to a coder, will be a flow chart. […] It is then the task of the coder to reduce this flow chart to coding, to a list in computer code [12, p. 1]

As "coder" never appeared in any report as an "intermediate point on the salary scale", and even a supervisor from Remington-Rand had to acknowledge that it was "more often the case than not [...] that the programmer and clerical coder are the same person" and simply omitted "the detailed flow chart" [15], why then did Hopper distinguish a separate coder? Because they could now be automated:

> It is this function, that of the coder, time-consuming and fraught with mistakes, that is the first human operation to be replaced by the computer itself. [12, p. 1-2]

Thus it was the introduction of "automatic coding" (sometimes also unfortunately called "automatic programming") that accounted for a retrospective, artificial distinction between the jobs. As she later wrote: "automatic coding comes into its own" [13, p.2] when "it can release the coder from most of the routine and drudgery of producing the instruction code. It may, someday, replace the

coder or release him to become a programmer. “ Again, the great advantage of automatic coding is given as: “the replacement of the coder by the computer”.
Many in the 1950s were far from convinced that automatic coding would be performant enough to compete with manual programming. But by introducing the distinction between a programmer and a coder - though it did not reflect the reality on the workfloor - Grace Hopper’s promotional talks made the idea of automatic coding more poignant.

4. *Conclusion*

The influence of powerful imagery and rhetorics in promotional material need not surprise us in computing. There is a long standing tradition of overselling new technologies and methods, claiming the next (industrial) revolution or promising a new technology that will outperform human beings. With the passage of time, however, it may become difficult to be aware of ideas and images that have acquired a life or their own and have become integrated as part of a historical narrative. As modern, digital and electronic computing is nearing its 100th anniversary, such retrospection does not become easier, though we may be in need of it more than ever before.
This particular case, where the praise of automatic programming conjured up the spectre of the coder, can be instructive for us today. There is a tradition that runs from Grace Hopper’s selling of “automatic coding” to today’s promises of large AI models such as Chat-GPT for revolutionizing computing by automating programming or even making human programmers obsolete altogether [19,20]. Then as now, it is certainly the case that the automatization of programming is ongoing and will upset or even redefine the division of labor in programming. However, this automatization is not a simple straightforward process that replaces the human element in one or more specific phases of programming by the computer itself, rather, many practices are locally embedded and seek out how new techniques can assist in the already existing tasks and jobs. Such transfers do not generalize easily, therefore, introducting such titles as “coders” or today’s “prompt engineers”, while easily grabbing attention, do not do justice to the time-consuming process of changing practice.

Bibliography

1. Abbate, J. Recoding Gender: Women’s Changing Participation in Computing, Cambridge, Massachusetts Institute of Technology Press, 2012.
2. Adams, C.W, Gill, S. and Combelic, D. Digital Computers: Advanced Coding Techniques. Massachusetts Institute of Technology, Summer Session 1954
3. Bullynck, M. Programming men and machines. Changing organisation in the artillery computations at Aberdeen Proving Ground (1916-1946). *Revue de Synthèse* 139 (3-4), 2018, 237-261.
4. Campbell-Kelly, M., Aspray, W., *Ensmenger*, N., & Yost, J.R. Computer: A History of the Information Machine (3rd ed.). New York: Routledge, 2014.
5. De Mol, L. and Bullynck, M. What's in a name? Origins, transpositions and transformations of the triptych Algorithm - Code – Program. In: Abbate,J. and Dick, S. (eds.). Abstractions and Embodiments of Computing. Johns Hopkins Press, 2022, 147-168.
6. Eckert, P.J. Weiner, Welsh, J.R.F., Mitchell, H.F. *The UNIVAC System. AIEE-IRE ’51: Papers and Discussions* presented at the December 10-12, 1951, *Joint AIEE-IRE Computer Conference*, 1951, 6–16.
7. Ensmenger, N. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise.* Cambridge, MA: MIT Press, 2010.
8. Goldstine, H.H. and von Neumann, J. *Planning and Coding of Problems for an Electronic Computing Instrument,* vol. 2 of *Report on the Mathematical and Logical Aspects of an Electronic*

Computing Instrument. Report in three parts prepared for US Army Ord. Dept. under Contract W-36-034-ORD-7481, 1947-48.

9. Grier, David Alan, When Computers were human, Princeton, Princeton University Press, 2005.

10. Haight, Thomas, Priestley, Mark, Rope, Crispin, ENIAC in Action, Cambridge (MA), MIT Press, 2016.

11. Harrison, J. O., Holberton, J. V., Lotkin M. Preparation of Problems for the BRL Calculating Machines. Ballistic Research Laboratories, Technical Note 104, Aberdeen Proving Ground, Maryland, 1949.

12. Hopper, G. *Automatic Programming—Definitions. Symposium on Automatic Programming for Digital Computers.* Office of Naval Research, Department of the Navy, Washington, DC, May 13-14, 1954: 1–5.

13. Hopper, G. Automatic coding for digital computers. a talk presented at the High Speed Computer Conference, Louisiana State University, 16 February 1955.

14. Holberton, B. Standard flow chart and coding conventions. Eckert-Mauchly Computer Corporation, 1950.

15. Rossheim, R.J. The Function of Automatic Programming' for Computers in Business Data Processing, Computers and Automation, 5 (2), pp. 6-9, 1956.

16. Stern, Nancy From Eniac to Univac: an Appraisal of the Eckert-mauchly Computers. Bedford, Mass., Digital Press, 1981

17. US Department of Labor, *Occupational Analysis Branch of United States Employment Service, Occupations in Electronic Data-Processing Systems.* Washington, 1959.

18. Weber, Philip H., Determing Salaries for Computer Personnel. Chicago, Office Appliance Co., 1960.

19. Welch, Matt. The End of Programming, Communications of the ACM, January 2023, 66 (1), 34-35.

20. Yellin, D. The Premature Obituary of Programming, Communications of the ACM, February 2023, 66 (2), 41-44.