The Mathematical Origins of Modern Computing

Mark Priestley

Abstract Modern computing has been shaped by the problems and practices of mathematics to a greater extent than is often acknowledged. The first computers were built to accelerate and automate mathematical labour, not as universal logical machines. Very specific mathematical objectives shaped the design of ENIAC, the first general-purpose electronic computer, and its successor, EDVAC, the template for virtually all subsequent computers. As well as machine architecture, software development is firmly rooted in mathematical practice. Techniques for planning large-scale manual computation were directly translated to the task of programming the new machines, and specific mathematical practices, such as the use of tables in calculation, profoundly affected the design of programs.

Key words: History of computing, ENIAC, EDVAC, programming, coding, subroutines.

1 Introduction

If there is a truth universally acknowledged in the history of computing, it is this: the "modern computer" was invented in the early 1940s and its design was first described in the *First Draft of a Report on the EDVAC* (von Neumann 1945b). In the preceding three years, a group at the University of Pennsylvania's Moore School of Electrical Engineering had designed and built ENIAC, a giant machine that among other things demonstrated the feasibility of large-scale electronic calculation. As ENIAC's design neared completion in 1944, the team began to plan a follow-up project, the EDVAC. They recruited the mathematician John von Neumann as a consultant, and in early 1945 he wrote a report describing, in rather abstract terms, the design of the new machine. This was the first systematic presentation of the new ideas, and proved highly influential. By the end of the decade the first machines

Mark Priestley, e-mail: m.priestley@gmail.com

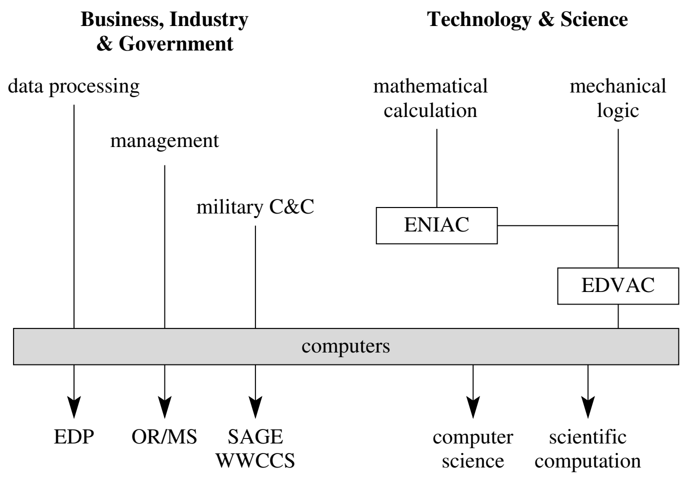

Fig. 1 The communities of computing (redrawn extract from Mahoney (2005), fig. 5, copyright © Institute of Materials, Minerals and Mining, reprinted by permission of Taylor & Francis Ltd, http://www.tandfonline.com, on behalf of Institute of Materials, Minerals and Mining.)

based on the EDVAC design were operational, marking the first step on a ladder of technological progress leading to the ubiquity of computational machinery today.

Historian Michael Mahoney (2005) challenged such machine-centric views of computer history. Mahoney urged historians to turn their attention to the history of computing, not just the technical history of computers. He further argued that the history of computing cannot be understood as a single unified narrative. The computer can be many things to different people, generating a multitude of diverse stories. Mahoney supported his argument by appealing to a particular view of the nature of the computer: while acknowledging that the first computers were built to perform scientific calculations, he believed that the machines based on the EDVAC design were something different, not just calculators but "protean machines" that could be bent to any task.

> But making it [the computer] universal, or general purpose, also made it indeterminate. Capable of calculating any logical function, it could become anything but was in itself nothing (well, as designed, it could always do arithmetic). (Mahoney 2005, 123)

A machine which is in itself nothing cannot have much of a history. Instead, Mahoney urged, historians of computing should tell the stories of how the machine was introduced to and transformed, and was itself transformed by, a wide range of existing "communities": the people involved in areas of application such as data processing, management, or military command and control systems (Fig. 1).

It is striking that, despite Mahoney's revisionist intentions, this schema retains a prominent place for the traditional origin story involving ENIAC and EDVAC. On Mahoney's account, EDVAC has a dual nature. On the one hand, it is a room-sized mathematical calculator, built for very specific purposes by a particular group of people. But at the same time it is an abstract machine, "in concept a universal

Turing machine". According to Mahoney, it is this second, spectral machine which moves between communities. Being universal and general-purpose, its potential for use in different fields can be taken for granted.

From this point of view, the computer's mathematical origins are little more than an historical curiosity. Mahoney followed logician-turned-historian Martin Davis (2000) in seeing the crux of the computer's evolution as being an injection of logic between ENIAC and EDVAC that turned a brute calculator into an ethereal logic machine with, incidentally, the capability to do "arithmetic".[1]

However, the idea that a new technology can transform many application areas is not the novelty that Mahoney seems to suggest, and does not depend on the universal nature of the technology being transferred, as two examples from the prehistory of computing illustrate. In the 1920s, Leslie Comrie began an extended investigation into the use of punched card machinery to support scientific calculation, work that was continued by Wallace Eckert in the USA. Similarly, Tommy Flowers took with him to Bletchley Park the experience that he had gained with electronic switching before World War 2 in the British GPO, and deployed it very effectively in the development of the Robinson and Colossus machines. In this perspective, the idea that the invention of the computer might give rise to different histories of adoption in different areas is simply another example of a regular historical pattern.

The computer remains a special case in its breadth of application, of course, and this is a fact that calls out for explanation. In response, Mahoney appealed to the modern computer's "protean" nature. But how does the computer come to have such a nature? The conventional answer to this is technological: the "stored-program" concept, itself said to be derived from Turing's description of a universal machine, is the particular feature that allows a single machine to perform an unlimited variety of tasks.[2] But there is an unsatisfying circularity in the suggestion that it is the "universal" logico-technical properties of the computer that make it inevitable that it will find universal application. It is more illuminating to start with a functional characterization: if the computer is a technology of automation, what was it intended to automate? In his proposal for the ACE, a machine to be built at the UK's National Physical Laboratory, Turing suggested an answer to this question:

> How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely carrying out orders given to it in a standard form which it is able to understand. (Turing 1946, 3)

The modern computer, in other words, is a machine that obeys orders. As a matter of historical contingency, the first such machines were developed to automate the specific processes involved in large-scale mathematical calculation. This is far from being the incidental detail that Mahoney suggests, however, and deeply affected

[1] Similar views were widely canvassed in connection with Turing's centenary celebrations in 2012. An alternative perspective, challenging the view that logic played a central role in the development of EDVAC, has been articulated recently by historians including Tom Haigh (2014) and Edgar Daylight (2015). See also Section 5, below.

[2] The stored-program concept has been discussed, and its usefulness as an analytical category critiqued, by Haigh et al. (2014).

the ways in which computers could be deployed in areas outside mathematics, as computer scientist Donald Knuth's comments on the problems of carrying out data retrieval with electronic computers illustrate:

> Computers have increased the speed of scientific calculations by a factor of 10^7 or 10^8, but they have provided nowhere near this gain in efficiency with respect to problems of information handling. [...] We shouldn't expect too much of a computer just because it performs other tasks so well. (Knuth 1973, 551)

The first half of this chapter describes the effects of the mathematical context of innovation on the ENIAC and EDVAC projects and the machines they developed. The computer's mathematical origins are reflected in more than just its physical characteristics, however. The modern computer automated a certain kind of human labour, that of following a plan of computation in a more or less mechanical way. Many of the established practices of manual calculation were transferred to the new machines and profoundly shaped the ways in which they were used. The second half of this chapter examines how two such practices, the social organization of large-scale computation and the use of mathematical tables, were translated into the context of the automatic computer and the consequences of this for the way the new task of programming was conceived.

2 The organization of large-scale calculation

In the 1790s, the French engineer Gaspard Riche de Prony embarked on a mammoth project to calculate a new set of tables of logarithmic and trigonometric functions (Grattan-Guinness 1990). The undertaking was industrial in scale, and to manage it de Prony employed the principle of the division of labour recently described by Adam Smith in *The Wealth of Nations*, first published in 1776.

De Prony divided his workforce into three sections. The first section consisted of a small number of leading mathematicians who derived the formulas that would be used to calculate the various functions. These formulas were passed on to a second section of skilled but less eminent mathematicians whose job was to work out how to calculate the values of the formulas using the method of differences.

An advantage of the method of differences was that it enabled the functions to be calculated using only the basic operations of addition and subtraction. The third section consisted of relatively unskilled labour, many of them hairdressers who had been made redundant by changing fashions after the French revolution. The workers of the third section carried out sequences of additions and subtraction as specified by calculating sheets prepared by the second section. Rough working was carried out on loose sheets of paper, and the results were transcribed onto the calculating sheets, which were then passed back to the second section for checking.

The third section had little scope for the exercise of judgement or initiative. As Smith had observed, the division of labour often broke a complex task down into activities that were simple enough to be mechanized. Fully aware of de Prony's approach, Charles Babbage took advantage of this when beginning the development of his first Difference Engine in the 1820s:

> If the persons composing the second section, instead of delivering the numbers they calculate to the computers of the third section, were to deliver them to the engine, the whole of the remaining operations would be executed by machinery. (Babbage 1822, 10)

More than a hundred years after Babbage, large-scale computation was still being organized along the lines pioneered by de Prony. David Grier (1998) has described the organization of the Math Tables Project (MTP), a Depression-era project aimed at providing jobs for unemployed office workers in New York. The work of the MTP was directed by a Planning Committee which "developed the mathematical methodology, and prepared the computing instructions" that were passed onto the Computing Floor Division. This consisted of two groups of trained mathematicians who could be trusted to work independently: the "Special Computing Unit", who among other responsibilities helped the project leaders to prepare the worksheets for the "Manual Unit", and the "Testing Section". The Manual Unit were unskilled workers who were trained to perform to perform specific operations, such as multiplication by a single digit. Their work was directed by the worksheets.

Desk calculating machines were widely used in the 1930s to perform arithmetical operations, including multiplication and division. As the MTP grew, it acquired numbers of second-hand calculators and the size of the Manual Unit shrank as its suitably qualified members were promoted to the Machine Unit.

From the French revolution right through to the mid-twentieth century, then, the organization of large-scale calculation took the form of a pyramid resting on the base of a large group of mathematically unsophisticated (human) computers. The computers were expected to perform individual arithmetico-logical operations, with or without mechanical assistance, and to closely follow a plan telling them what operations to perform, in what order, and how and where to record the results. The ability to work independently and the exercise of initiative or judgement were not required.

It was precisely these characteristics that machine developers of the early 1940s were hoping to automate and that George Stibitz, designer of an influential series of machines at Bell Telephone Laboratories, made the defining property of computers understood as machines rather than human beings.[3]

> By "calculator" or "calculating machine", we shall mean a device (mechanical, electrical or what not) capable of accepting two numbers, A and B, and of forming some or any of the combinations $A+B$, $A-B$, $A\times B$, A/B. By "computer", we shall mean a machine capable of carrying out automatically a succession of operations of this kind and of storing the necessary intermediate results". (Stibitz 1945, 1–2)

In the 1830s Babbage had made the first attempt to design such a computer with his work on the Analytical Engine. Around a hundred years later, Konrad Zuse in

[3] The word "computer" before 1945 did not always refer to a human being. From the 1890s onward, "computers" were also computational aids, sometimes booklets containing useful collections of tables and methods (Hering 1891), but more often special-purpose circular slide-rules embodying particular formulas or algorithms (Halsey 1896). David Mindell (2002) has traced the further usage of the word in the 1930s in the field of fire-control systems in the US military.

Germany and Howard Aiken in the USA independently began projects leading to the construction of the first machines capable of automatically carrying out a sequence of operations.

3 Automating calculation

In 1935, Zuse set up a workshop in his parents' Berlin apartment and began work. The following year, he submitted a patent application describing a machine which would automatically execute "frequently recurring computations, of arbitrary length and construction, consisting of an assembly of elementary arithmetic operations" (Zuse 1936, 163). The operations to be performed were described by what Zuse called a "computation plan" which would be presented to the machine in some suitable form, such as a punched tape. As an example, Zuse gave a plan for calculating the determinant of a 3×3 matrix. This involved a total of 17 operations, each with two operands: 12 multiplications, two additions and three subtractions.

Zuse developed a series of machines designed along these lines. The third of these machines, the Z3, was completed in 1941 and is now considered to be the first programmable computer. The Z3 and its predecessors were destroyed in air raids, but Zuse's next machine, the Z4, survived and was moved to Zurich, where it played an important role in the post-war development of European computing.

In 1937, Harvard graduate student and physics instructor Howard Aiken wrote a proposal for "an automatic calculating machine specifically designed for the purposes of the mathematical sciences" (Aiken 1937). He observed that existing punched-card calculating machinery did "the reverse of that required in many mathematical operations", in that it allowed the evaluation of a limited range of formulas on sequences of data read from punched cards. By contrast, Aiken believed that the characteristic of scientific calculation was that it required long and varied sequences of operations to be carried out on relatively small amounts of data. In principle, this could be done on existing machinery by manually switching from one operation to another: it was precisely this manual switching that Aiken planned to automate.

Aiken managed to enlist the help of IBM in building his machine, officially called the IBM Automatic Sequence Controlled Calculator; it later became widely and more conveniently known as the Harvard Mark I. On its completion in 1944, IBM donated the machine to Harvard, where it ran for many years, initially under the control of the US Navy.

Every aspect of Mark I was determined by its role in mathematical calculation. Like Zuse's machines, it was equipped with a number of general purpose registers, or counters, which stored results and allowed them to be retrieved when needed. Aiken explained the need for storage registers as a consequence of the pragmatics of conventional mathematical notation:

> The use of parentheses and brackets in writing a formula requires that the computation must proceed piecewise. [...] This means that a calculating machine must be equipped with

> means of temporarily storing numbers until they are required for further use. Such means are available in counters. (Aiken 1937, 198)

The counters stored incoming numbers by adding them to their existing contents, thus enabling Mark I to carry out addition in general. Subtraction was carried out using complements. There were specialized units for multiplication and division, to compute the values of selected exponential and trigonometric functions, and to interpolate between values read from a paper tape. But the heart of the machine was the sequence mechanism. This read a list of coded instructions that had been punched onto a paper tape and invoked the corresponding operations. By simply changing the tape, Mark I could be instructed to carry out any desired computation.

4 The structures of computation

The sequence of operations performed by the Z3 or Mark I was determined by the sequence of instructions read from the machines' tapes. To evaluate a simple formula, the tape would simply contain one instruction for each operation that the machine was to execute, but this approach did not scale up well to more complex problems. Many calculations have an iterative structure in which a small sequence of operations is repeatedly performed. It would be wasteful to punch a tape with the same instructions over and over again, and in many cases this would not even be possible. In general, a mathematician cannot tell in advance how many iterations of the operations will be required and instead has to rely some property of the results obtained so far to determine when the calculation should stop.

The conditional branch instructions of modern programming languages address these issues by allowing computations to diverge when necessary from the default sequence of instructions. The earliest computers did not have branch instructions, however, and various ad hoc approaches were adopted instead. Babbage proposed mechanisms to "back up" the Analytical Engine's cards so that instructions could be repeated, while Mark I's tapes were made "endless" by gluing one end to the other. An endless tape would loop indefinitely, carrying out the instructions on it over and over again. To interrupt a loop, Mark I had a conditional instruction that stopped the machine when, say, the results obtained so far reached certain limits of tolerance, but in order to continue with the next stage of the computation a new tape had to be mounted by the operators and the machine restarted. The first machine to fully automate computation, allowing loops and conditional branches to be freely utilized, was ENIAC.

ENIAC was the brainchild of a physicist, John Mauchly, who had taken up wartime employment at the Moore School.[4] The school had a long-standing collaboration with the Army Ordnance Bureau's proving ground in Aberdeen, in nearby Maryland, and in particular with its Ballistics Research Laboratory (BRL), an important centre of calculation in the interwar years. BRL had supported the Moore

[4] The following two sections draw extensively on the material in (Haigh et al. 2016).

School's acquisition of a differential analyzer, with the understanding that in the event of war it would be made available for BRL's use. Developed by Vannevar Bush (1931) at MIT, the analyzer was a cutting-edge analogue machine which used mechanical integrators to solve differential equations.

When war broke out, BRL faced the challenging task of compiling firing tables for a vast range of new ordnance and ammunition. These tables integrated large amounts of experimental data and ballistic calculation, and told gunners how to aim their weapons to hit a specific target. To compile a table, many trajectories—the predicted paths of projectiles fired from the gun—had to be calculated, each requiring the solution of a set of differential equations that could take a human computer several hours. Invoking the terms of the earlier agreement, BRL set up a satellite computing centre at the Moore School overseen by Herman Goldstine, a mathematician whose wartime commission had seen him posted to BRL. Goldstine and his wife Adele were responsible for training and supervising teams of computers calculating trajectories. The Moore School's differential analyzer was extensively used in these calculations.

Mauchly was not directly involved in the firing table work, but he supervised a group carrying out manual computation and was familiar with the design and use of the analyzer. He had a long-standing interest in the use of electronics for calculation, and in August 1942 brought these interests and experience together in the form of a brief proposal for an electronic analogue of the differential analyzer. He estimated that the use of "high-speed vacuum tubes" would allow trajectories to be calculated in a fraction of the time taken by the mechanical analyzer, let alone by manual calculation. The proposal eventually came to the attention of Herman Goldstine, who saw great potential in it. Mauchly and Presper Eckert, a talented electronic engineer who had trained Mauchly when he first arrived at the Moore School, wrote a more detailed outline and a collaboration was soon agreed whereby the Moore School would build an electronic machine for BRL.

Although it was envisaged that the machine would spend a lot of its time calculating trajectories, its design was not limited to that particular application. As Mauchly had explained:

> There are many sorts of mathematical problems which require calculation by formulas which can readily be put in the form of iterative equations. [...] Since a sufficiently approximate solution of many differential equations can be had simply by solving an associated difference equation, it is to be expected that one of the chief fields of usefulness for an electronic computor [sic] would be found in the solution of differential equations. (Mauchly 1942)

Mauchly's use of the phrase "electronic computer" seems very natural to modern-day readers, but would have been quite unfamiliar in 1942. Mauchly had described the new machine as an "electronic difference analyzer", but "computer" was soon added to the machine's name to reflect its potential generality, as Grist Brainerd, the Moore School academic in charge of the project, explained:

> The electronic difference analyzer and computer is a proposed device never previously built, which would perform all the operations of the present differential analyzers and would in

> addition carry out numerous other processes for which no provision is made on present analyzers. It is called a "difference" analyzer rather than a "differential" analyzer for technical reasons. (Brainerd 1943)

The new machine soon became terminologically independent of its predecessor, being dubbed the "Electronic Numerical Integrator and Computer", or ENIAC.[5] The numerical solution of differential equations by iterative means became ENIAC's signature application, but over the course of its working life it was applied to a much wider range of calculations than simply trajectories. Nevertheless, as late as the early 1950s, "artillery and bomb ballistics computation" made up a quarter of its workload (Reed 1952).

Mauchly may have used the term "computer" to emphasize that ENIAC, unlike a simple calculator, would be automatically sequenced and, like a human computer, able to work independently. In this respect, electronic speed was problematic, as it meant that the familiar technique of reading coded instructions from paper tape was simply too slow. Instead, the team adopted what they later described as a stop-gap solution in response to the urgency of a wartime project and designed ENIAC as a collection of specialized calculating units. They shared with Zuse and Aiken the view that calculations could be specified as sequences of instructions, but they adopted a different technological approach to realizing the instruction sequences. Instructions were set up on "program controls" on each unit, and computations were sequenced by cabling these controls together in problem-specific configurations. As it turned out, this gave ENIAC a flexibility that allowed a greater degree of automation than was possible on the tape-controlled machines.

The ENIAC team delivered their first progress report at the end of 1943, six months after the start of the contract funding the project. After extensive research into the existing state of the art, a new design for the machine's basic electronic counters had been decided on, but nothing had been constructed apart from a few test circuits. Plans for some units were fairly well advanced, but others had barely been started. There were many open questions about the design of the machine, and it had not yet been demonstrated that large numbers of unruly electronic valves could be persuaded to collaborate reliably and work as required.

Despite the provisional and incomplete state of the hardware design, however, the report was accompanied by detailed plans showing how ENIAC could be set up to calculate a ballistic trajectory.[6] Acting as a kind of feasibility test, this exercise enabled the team to settle many aspects of ENIAC's design. Different algorithms were investigated, the choice between them being governed by a variety of practical considerations. Would the numerical properties of the equations allow a reasonable degree of accuracy to be preserved throughout the calculation? Would the number of operations to be carried out and intermediate values to be stored physically fit onto

[5] In a 1962 affidavit, Brainerd recalled that Paul Gillon of the Ordnance Bureau, an enthusiastic supporter of the project, came up with the new name.

[6] Haigh et al. (2016) attributed these plans to Arthur Burks. Subsequent archival research suggests that the work was in fact split between Burks and Adele Goldstine, with Goldstine taking the lead on the mathematical analysis of the problem, expressed in a "setup form", and Burks mapping it onto ENIAC's distributed programming system in the form of a "panel diagram".

ENIAC? Analysis by the Moore School mathematician Hans Rademacher showed that the relatively unfamiliar Heun method would be suitable, and the problem was reduced to a set of 24 simple difference equations. This analysis also enabled the size of ENIAC's accumulators to be fixed: numbers had to be stored to a precision of ten decimal digits to enable the computed results to be sufficiently accurate for BRL's purposes.

The analysis of the structure of the computation was just as significant as the numerical work. The trajectory calculation was split into four basic sequences of instructions: setting up the initial conditions, performing an integration step, printing a set of results, and carrying out a check procedure. These sequences were combined in a complex structure which included two nested loops: after the initial sequence was complete, a loop would print a set of results and carry out the check procedure 40 times; each set of results was calculated by performing the integration step 50 times.[7] In 1943, the team had little idea how a computation of such complexity would be controlled, and proposed a unit called the "master programmer" which would control the repetition of instruction sequences and move from sequence to sequence when required.

In the following months, the team set to work on the master programmer. Central to its design was a multi-functional device known as a "stepper" which controlled the initiation of up to six program sequences, one after the other. Each stepper had a counter associated with it to keep track of how many times the current sequence had been executed. Once a sequence had been repeated a specified number of times, the stepper would move the machine on to the next sequence. Conditional control was enabled by routing pulses derived from the results already calculated into a special "direct input" socket which advanced the stepper independently of the number of repetitions that had been counted.

ENIAC, then, was designed to solve a specific type of mathematical problem, but it had to be able to do so completely automatically: if its operators had to change instruction tapes, as on Mark I, the advantages of its electronic speed would be lost. The analysis of the trajectory calculation revealed the level of control flexibility required and led to the design of the master programmer, a device capable of controlling highly complex computations built up using the fundamental structures of loops and conditional branches. As a result, ENIAC was capable of tackling a wide range of problems, although in practice physical constraints such as its small amount of high-speed storage limited its scope of application (Reed 1952).

[7] ENIAC would therefore carry out 2000 integration steps to calculate a single trajectory, many more than was feasible in a hand calculation. This was one reason why the numerical properties of the method to be used had been studied so closely: with many more arithmetical operations being carried out, errors could be expected to accumulate more rapidly.

5 The computer as mathematical instrument

By the summer of 1944, ENIAC's design was virtually complete and the team were beginning to think about the future. Anxious to secure a new contract before the generous wartime funding arrangements dried up, they proposed to BRL's director, Colonel Leslie Simon, a new research and development project for a machine which would address two perceived weaknesses of ENIAC's design: its paucity of high-speed storage, and the time-consuming way in which problems were set up.

At around the same time, John von Neumann discovered ENIAC. Despite the fact that he had been a member of BRL's Scientific Advisory Committee since 1940, he only found out about the machine, according to Herman Goldstine, thanks to a chance meeting at Aberdeen railway station. Within a month of this meeting, however, an agreement had been reached for a contract to develop a new machine. Historians have speculated about von Neumann's role in helping to bring about this decision, but one likely consequence of his involvement was to establish just what the new machine was for. In 1943, BRL had a very clear idea of why they needed ENIAC: they faced a bottleneck in the calculation of firing-table trajectories, and the need to address this requirement shaped ENIAC's design in many ways. By contrast, in a context where it was cutting back on long-term research projects, the Bureau of Ordnance might not have been so keen to support a proposal framed in terms of the need to address shortcomings in a machine it was still in the process of paying for. As Babbage had discovered a century earlier, this is not a great strategy for winning a funding body's support.

Matters moved quickly. On August 29, at a meeting attended by both Goldstine and von Neumann, BRL's Firing Table Reviewing Board decided to support a new contract with the Moore School, to develop "a new electronic computing device". The Board minuted that the new machine would be "cheaper and more practical to maintain" than ENIAC, would be able to store large quantities of numerical data, and would be easy to set up for new problems. The Board also noted that the new machine would be "capable of handling many types of problems not easily adaptable to the present ENIAC" (see Haigh et al. 2016, 134).

Von Neumann brought to the meeting the perspective of a user, not a computer builder. Although he proved more than capable of engaging with the gritty details of vacuum tubes, he was also engaged in a continent-wide search for raw computing power for a variety of projects, including the Manhattan Project at Los Alamos. In March he had used the IBM punched card machines at BRL to carry out some test calculations on hydrodynamical shock problems, noting that:

> The actual computations on each problem required 6-12 working hours net, and the entire program (setting up, etc), insofar as these three problems were concerned, took less than ten days. [...] In the truly many-dimensional cases the possibility of using other types of machines will also have to be investigated. (von Neumann 1944b, 375, 379)

The possibility of using ENIAC for similar work was quickly investigated. By August 21, as Goldstine reported:

> Von Neumann is displaying great interest in the ENIAC [...] He is working on the aerodynamical problems of blast and runs into partial differential equations of a very complex character. By greatly simplifying his equations he is able to get a one dimensional equation that is solvable in four hours on the IBM's. We calculate that it will take ten seconds on the ENIAC counting the printing time. (Goldstine 1944b)

But not even ENIAC was powerful enough. The day after deciding to support the new contract, the Firing Table Reviewing Board sent a detailed memo to Simon outlining the rationale for their decision. Since the new machine would be more flexible and capable of storing large amounts of numerical data:

> It would make possible the solution of the complete system of differential equations of exterior ballistics [...] these equations are too complicated in character to be handled by the differential analyzer, the Bell Telephone machines, the IBM machines, or the present ENIAC in a reasonable length of time. (BRL 1944)

The Board also noted the application of the new machine to the "extensive and unusual computations" needed to make use of the data produced by BRL's new wind tunnel. Existing machines, including ENIAC, would be "most useful in extensive but less complicated routine calculations". The wind tunnel played a prominent role in selling the new proposal to BRL and its paymasters. In mid-September Brainerd wrote to Colonel Paul Gillon of the Bureau of Ordnance referring to:

> some rather extensive discussions concerning the solution of problems of a type for which the ENIAC was not designed. [...] Dr. Von Neumann is particularly interested in mathematical analyses which are the logical accompaniment of the experimental work which will be carried out in the supersonic wind tunnels. Unfortunately practically all of these problems are tied up in non-linear partial differential equations, the solutions of which is is impractical to obtain with any known equipment now existing or being built. (Brainerd 1944a)

Brainerd was now careful to suggest that ENIAC's perceived shortcomings were not defects, but rather adaptations to the particular type of problem it was designed to solve. These representations evidently had the required effect: towards the end of October, a supplement to the ENIAC contract was signed authorizing a nine-month contract on "an Electronic Discrete Variable Calculator", starting on January 1, 1945.[8] Von Neumann's existing role at BRL was expanded, allowing him to act as a consultant to the Moore School for the new project (Goldstine 1944a).

Simon received yet another memo on the subject, this time from von Neumann himself, in January 1945. Von Neumann emphasized the importance of "general aerodynamical and shock-wave problems" and the need to make "full and efficient use of the Supersonic Windtunnel", and he pointed out that ENIAC and the Bell Labs machines were not really suited to the kind of calculations required:

> The differential equations are usually partial and 2 or 3 dimensional, and they are therefore in the simplest cases just on the margin of what the present equipment can handle, and in all other cases far outside its compass. [...] The EDVC [sic] is being designed with just this type of problem in view. (von Neumann 1945a)

[8] Brainerd (1944b) described the machine thus in a memo to the Bureau of Ordnance. By the time the project's first progress report was issued, at the end of March 1945, it had firmly acquired the acronym EDVAC, in which the C stood for "computer".

In the latter part of 1944, then, a rather vague aspiration to build a machine that would address some of ENIAC's shortcomings was refined into a proposal for a computer optimized to solve a class of problems of pressing concern to BRL, multi-dimensional, non-linear, partial differential equations. Brainerd was quick to spell out the connections between this application and the team's technical ambitions:

> If a two-dimensional problem is to be solved [...] many thousands of values of quantities must be stored while the process is being carried on. It is on this point of the great amount of storage capacity required that existing and contemplated machines fall down. There is also a further point that the programming of the carrying out of the solutions is far more complicated than permitted by existing or contemplated machines. (Brainerd 1944a)

EDVAC, then, needed a large high-speed store because the calculations it was being built to carry out generated large amounts of numerical data. But this also suggested a solution to the problem of setting up the machine quickly:

> To evaluate seven terms of a power series took 15 minutes on the Harvard machine of which 3 minutes was set up time, whereas it will take at least 15 minutes to set up ENIAC and about 1 second to do the computing. To remedy this disparity we propose a centralized programming device in which the program routine is stored in a coded form in the same type storage devices [sic] suggested above [to hold numerical data]. (Goldstine 1944c)

All previous automatic computers had used different storage media for numbers and program instructions: numbers were stored in counters of various kinds, while instructions were read from paper tape or, in the case of ENIAC, set up on dedicated pieces of hardware. If instructions were to be available at electronic speed, they could not be read when needed from an external medium, but had to be placed on the machine before the computation began. As Goldstine noted, a new device—mercury delay lines—had been proposed for the cost-effective storage of large amounts of numerical data. If instructions were coded as numbers, as on the Harvard and Bell Labs machines, it would obviously be possible to use the same kind of device to hold the instructions.

At this point, Goldstine's proposal was that, instead of using different media to store numbers and instructions, they could be held in storage devices of the same type. What is often taken to be a defining characteristic of the modern computer, storing data and instructions in a *single* device, was adopted rather gradually. In the *First Draft*, after carefully listing all the different types of information that EDVAC would have to store, von Neumann commented:

> While it appeared, that various parts of this memory have to perform functions which differ somewhat in their nature and considerably in their purpose, it is nevertheless tempting to treat the entire memory as one organ, and to have its parts even as interchangeable as possible for the various functions enumerated above. (von Neumann 1945b, 6)

Some problems needed lots of programming instructions but used little numerical data, while others were exactly the reverse. As Eckert explained the following year, a single store would give EDVAC valuable flexibility:

> Aside from simplifying the construction of the machine, this move eliminates for the designer the problem of attempting to find the proper balance between the various types of memory [...] The proper subdivision of the memory, even for a restricted set of problems,

such as the ENIAC is designed to handle, is too variable from problem to problem to permit an economical compromise. (Eckert 1946, 112)

However, the code proposed in the *First Draft* clearly distinguished numbers and instructions, and treated the two kinds of data rather differently. EDVAC's memory would still be explicitly partitioned, recreating on a problem-by-problem basis the separate storage devices that Goldstine envisaged.

The tape of Alan Turing's universal machine of 1936 also held both data and coded instructions, a fact that has led some writers to suppose that there is a simple "stored program" concept, invented by Turing and subsequently implemented by the new machines of the mid-1940s. The complexities and confusions surrounding the term "stored program" have been analysed by Haigh et al. (2014), and it is sufficient here to note that EDVAC's unitary memory was not the result of the application of an insight drawn from mathematical logic, but of a series of pragmatic engineering decisions taken during the design of a machine requiring an unprecedentedly large store in order to address a particular class of mathematical problem.

A more significant innovation of the *First Draft* was to give programs the ability to modify their own instructions in certain ways as computations progressed. This had profound consequences for EDVAC's mathematical capabilities, making it feasible to write programs that operated on large vectors and matrices, not just on a small number of individual variables. Without this, the machine would not have been able to solve the partial differential equations of interest to von Neumann. There is nothing like this in Turing's earlier logical work.

Like ENIAC, then, EDVAC was designed and sold to its sponsor as a mathematical instrument with a rather specific purpose. Designing a machine capable of carrying out the required calculations led to a number of features that that are now considered definitional of the modern computer, such a large unitary memory and code that allows programs to modify their own instructions. There is no need to postulate an "injection of mathematical logic" in order to explain the origins of the computer and little, if any, evidence in the archival record of such an injection.

6 Planning and coding

The last two sections have examined some of the ways in which the computer's mathematical origins shaped its technological design. The influence of mathematics was not limited to hardware, however: the following sections will explore how the work practices within which automatic computers were situated profoundly affected early conceptions of computer programming.

The ENIAC progress report issued at the end of 1943 positioned the plans for the trajectory calculation in the context of a "general setup procedure" consisting of three phases. The first phase was mathematical, involving "the reduction of the given set of equations or relations to such a form that they can be solved by the ENIAC" (Moore School 1943, XIV (1)). This involved transforming the equations so that they only used the basic operations provided by ENIAC and ensuring that the computation would fit within the limits of its hardware, both in terms of the number of operations involved and the accuracy of the results that would be obtained. For the trajectory calculation, this phase resulted in a set of difference equations allowing a numerical solution of the equations of exterior ballistics to be calculated.

The second phase involved mapping these difference equations onto ENIAC's hardware. Variables were assigned to accumulators, and decisions were taken about numerical matters such as the number of decimal places and the position of the decimal point. Once this was done:

> this phase of setup reduces to the somewhat routine task of scheduling the operations and the corresponding connections. There are many possible arrangements for each problem, however, so that some skill is involved in chasing a suitable and preferred one. (Moore School 1943, XIV (2))

The results of this phase were given in a "setup form" describing the sequencing of the operations and the numerical details, and a "panel diagram" giving details of exactly how switches would be set and connections plugged so that ENIAC would perform the operations in the required order. Plans for the trajectory calculation were attached to the report. Based on the information in the panel diagram, the third phase of the procedure was rather more routine, involving "the manual plugging in of the various conductor cables and the manual setting of the various program switches" (Moore School 1943, XIV (3)).

The report claimed no originality for this three-phase procedure, pointing out its similarity to the way equations were set up on the differential analyzer. But its roots go back much further than that: the three phases correspond quite clearly to the basic division of labour devised by de Prony in the eighteenth century. The first phase, putting "the given equations [...] in a form suitable for the machine", corresponds to the work of the mathematicians in de Prony's first section, and Adele Goldstine and Arthur Burks, developing detailed computational plans that used only basic arithmetical operations, would have been natural members of the second section. The most significant change is a direct consequence of automation: the arithmetical hairdressers of de Prony's third section have been replaced by ENIAC, and human labour is only required to set up ENIAC to perform the calculation and supervise it while in operation.

The similarity extends even to the checking of calculations. De Prony had called for calculations to be carried out in two different ways, with the workers of the second section checking that the results were consistent. Every time ENIAC printed a set of results, a single integration step would be carried out with known data and the results printed. These would be checked by the operators, and any discrepancies with the expected results would indicate that a hardware fault had occurred.

Howard Aiken's group at Harvard employed a similar division of labour when preparing computations for Mark I. The first step was taken by "the mathematician who chooses the numerical method best adapted to computation by the calculator" (Harvard 1946, 50). Relevant factors considered at this stage included the accuracy and the speed of the calculation, and also the ease with which it could be checked.

The chosen method was then expressed in terms of Mark I's basic operations. A variety of notations were used at this stage. Coding sheets (Harvard 1946, 49) were used to define the basic sequence of operations to be punched onto instruction tapes, and diagrams were prepared showing how to wire the plugboards that some of Mark I's more complex units possessed.

Mark I was not fully automatic, however, and its operators were a integral part of computations, being required, for example, to change tapes when necessary. As well as coded instructions for Mark I, therefore, detailed operating instructions had to be drawn up for each calculation. In this context, the equivalent of de Prony's third section was the cyborg assemblage of Mark I and its operators. The Harvard group preserved the traditional status distinctions between the sections: operators were "enlisted Navy personnel" (Bloch 1999, 87), whereas the mathematicians were civilians or commissioned officers.

Historians have sometimes described the origins of programming as a secondary process that followed the development of the computing hardware. For example, Nathan Ensmenger (2010, 34) writes that programming was "little more than an afterthought in most of the pioneering wartime computing projects". At least in the case of ENIAC and EDVAC, this is not true: detailed plans, or programs, were prepared as part of the design process in both projects and directly influenced central aspects of the machines' design, such as ENIAC's master programmer.

It would be more accurate to say that the participants in these wartime projects did not view programming as being something particularly novel or problematic. Machines were built to carry out specific mathematical tasks and their designers assumed that existing well-understood procedures for planning and organizing large-scale calculations could be straightforwardly applied to the new situation. Moving from human to automatic computation led to changes in the way that the accuracy of calculations was estimated and their results checked, but the overall workflow of the planning process was unchanged. The biggest difference was that instead of handing a computation sheet to a human, the instructions it contained had to be translated into machine-readable form, but once the sequence of low-level operations had been decided on, this was thought to be a straightforward procedure. The changes brought about by automation were localized at a late stage in the overall planning process, as von Neumann pointed out when preparing a "tentative computing sheet" for a Monte Carlo simulation. It was, he said,

> neither an actual "computing sheet" for a (human) computer group, nor a set-up for the ENIAC, but I think it is well suited to serve as a basis for either. (von Neumann 1947, 152)

In the first of an influential series of reports on *Planning and Coding of Problems for an Electronic Computing Instrument*, Goldstine and von Neumann (1947) gave a detailed account of how existing practices of large-scale calculation could be adapted for use with automatic computers. Although the word "programming" was being used in its modern sense as early as 1944,[9] Goldstine and von Neumann chose not to use it. Instead, they split the overall workflow into the two major phases of "planning" and "coding". The division between the two phases marked the point at which techniques specific to automatic computers became important.

[9] Goldstine was an early adopter of the terminology; see, for example, the uses of "programming" and "program routine" by Brainerd (1944a) and Goldstine (1944c) quoted in the previous section.

Goldstine and von Neumann described planning as a “mathematical stage of preparations”. Echoing the approach taken by the ENIAC and Mark I designers, they explained that planning involved developing equations to model the problem at hand, reducing these to “arithmetical and explicit procedures”, and estimating the “precision of the approximation process”. They emphasized that all three steps in the planning stage were “necessary because of the computational character of the problem, rather than because of the use of a machine” (Goldstine and von Neumann 1947, 19).

The coding phase was less familiar and so discussed in much more detail. It was divided into two stages. A “macroscopic” stage corresponded to the second phase of the ENIAC setup procedure. It began by expressing the structure of the program in diagrammatic form, using the new flow diagram notation that Goldstine and von Neumann had developed, and drawing “storage tables” summarizing the data used by the program. The subsequent “microscopic” stage corresponded more closely to what in understood by “coding” today, and involved expressing the contents of the various boxes in the flow diagram in machine code. Some routine manipulations of the code were then carried out to turn it into its final machine-readable form.

By 1948, two further *Planning and Coding* reports, containing a number of worked examples, had been issued. The reports were highly influential and the flow diagram notation was widely adopted. Ensmenger (2016) has pointed out that as programming industrialized, flow diagrams came to function as boundary objects, notations inhabiting “multiple intersecting social and technical worlds” and flexible enough to enable communication between groups as disparate as managers, system analysts and programmers. Initially, however, they sat on the boundary between the planning and coding stages of the program preparation process. As computers came to be used for tasks that were not exclusively mathematical, or where a “mathematical stage of preparation” became less applicable, development began with a stage of “analysis” whose results, documented as a flow diagram, became the input for the more machine-oriented aspects of the workflow.

As experience with the new machines was gained, it quickly became apparent that planning and coding was not quite as straightforward as expected. The exercise of preparing instructions for a machine revealed the extent to which planners had relied on the humans of the third section to display intuition and common sense, even when they were supposedly acting “mechanically”. The English mathematician Douglas Hartree, one of ENIAC’s first users, commented on a typical breakdown in an automated calculation:

> A human computor, faced with this unforeseen situation, would have exercised intelligence, almost automatically and unconsciously, and made the small extrapolation of the operating instructions required to deal with it. The machine without operating instructions for dealing with negative values of z could not make this extrapolation. (Hartree 1949, 92)

The moral that Hartree drew from this experience was that programmers needed to take a “machine’s-eye view” of the instructions being written, and this blurring of the boundaries between human and machinic agency is nicely captured in the image of the human “automatically” exercising intelligence. However, it was more common to call for a more exhaustive and rigorous planning process. In what is often described as the first programming textbook, the Cambridge-based team of Maurice Wilkes, David Wheeler, and Stanley Gill explained that:

> A sequence of orders [...] must contain everything necessary to cause the machine to perform the required calculations and every contingency must be foreseen. A human computer is capable of reasonable extension of his instructions when faced with a situation which has not been fully envisaged in advance, and he will have past experience to guide him. This is not the case with a machine. (Wilkes et al. 1951, 1)

As this indicates, Goldstine and von Neumann's view of computer programming as a form of planning quickly became standard. The first challenge to the perceived limitations of this approach would not emerge until the mid-1950s, a development outlined in the final section of this chapter.

7 From tables to subroutines

The influence of mathematical practice on the use of automatic computers is visible not only in the organization of complete computations, but also in the details of specific programming techniques. An interesting example of this is the relationship between the use of tables in manual computation and the development of the idea of the subroutine.

The use of tables was so engrained in mathematical practice that the Harvard Mark I's designers put it on a par with the familiar operations of addition, subtraction, multiplication and division, writing that the machine was designed to carry out computations involving "the five fundamental operations of arithmetic": the fifth operation was described as "reference to tables of previously computed results" (Harvard 1946, 10). Tables were a ubiquitous feature of manual computation. A typical table would hold the precomputed values of a function, and when a value was required the (human) computer would interrupt work on the main calculation, take the appropriate volume of tables down off the shelf, look up the required value, and copy it into the appropriate place on the worksheet. Interpolation was used to obtain values for arguments that fell between those printed in the table.

Mark I contained dedicated hardware to support each arithmetic operation. Table look-up was implemented by three "interpolation units". These units read numerical data from tapes containing equally spaced values of the function argument, each followed by the coefficients to be used in the interpolation routine (Harvard 1946, 38, 47). When a function value was required, the argument was sent to an interpolation unit. The unit would then search the tape for the appropriate value of the argument, read the interpolation coefficients, and carry out a hardwired routine to calculate the required value.

Mark I also had special-purpose units to compute logarithms and values of the exponential and sine functions. Unlike the interpolators, these units did not read a tape, but executed a built-in algorithm to compute the required values directly. Nevertheless, the units were described as "electro-mechanical tables" (Harvard 1946, 11), a terminological choice that makes clear that Mark I's designers were not only transferring the use of mathematical tables in manual computation into the world of automatic machinery, but also using the experience of the past as a way of making sense of the new machine.

ENIAC's designers also considered table look-up to be one of the machine's basic capabilities (Moore School 1943, XIV (1)), and took a similarly explicit approach to supporting the use of tables. Numerical information was stored on three "portable function tables", large arrays of switches on which a table of around 100 values could be set up, indexed by a two-digit argument. This data was read by a "function table" unit, the whole arrangement being optimized to make it convenient to read the five values required for a biquadratic interpolation. Unlike Mark I, however, ENIAC had no dedicated interpolation unit. It was left to the user to set up an interpolation routine suitable for the problem at hand, and many examples of such routines are presented in Adele Goldstine's 1946 manual and other reports.

There is a tension apparent in Mark I and ENIAC between the alternatives of looking up tabular data and computing values when needed. While mathematical functions could be computed on demand, some applications, such as calculating a trajectory, made use of empirical data for which no formula was available. There was no alternative to storing such tables explicitly. The volume of tabular data to be stored was one of the issues that the EDVAC team considered when estimating the size of memory the machine would need, and von Neumann summarized the situation as follows:

> In many problems specific functions play an essential role. They are usually given in form of a table. Indeed in some cases this is the way in which they are given by experience [...], in other cases they may be given by analytical expressions, but it may nevertheless be simpler and quicker to obtain their values from a fixed tabulation, than to compute them anew (on the basis of the analytical definition) whenever a value is required. (von Neumann 1945b, 4-5)

He suggested that common functions such as log, sin and their inverses could be treated by table look-up rather than calculation. Interestingly, Mark I's designers had made precisely the opposite choice, providing the dedicated electromechanical tables to compute the values of these elementary functions on demand.

Large computations would typically have to look up many values, and so perform multiple interpolations. On Mark I, this would simply require repeated calls to the interpolation units, but the situation was a bit more complicated on ENIAC where the interpolation routine was set up by the programmer. Clearly, setting up the instructions repeatedly would be a wasteful and ultimately infeasible approach. To perform multiple interpolations, the designers had to find a way to return to a different place in the main instruction sequence each time the interpolation routine was carried out. This capability was provided by the versatile steppers, the key components of ENIAC's master programmer. The mid-1944 progress report explained how this could be done, making the connection with interpolation explicit:

> Thus within a given step of integration a certain interpolation process may be used several times. This sequence need be set up only once; by means of a stepper the same sequence can be used whenever needed. (Moore School 1944, IV-40)

This idea of “computation on demand” was naturally soon generalized, and it was recognized that it would be useful to be able to easily reuse any sequence of instructions, not only those computing familiar mathematical functions. In August, 1944, von Neumann reported to Robert Oppenheimer on the progress of the Bell Labs machine. Like Mark I, this machine would read instructions from paper tape, but unlike the Harvard machine, it would have more than one sequence unit. As von Neumann (1944a) noted, it would employ “auxiliary routine tapes [...] used for frequently recurring sub-cycles”. There is no suggestion that these auxiliary tapes would be limited to the purpose of interpolation or table look-up.

This turned out to be an issue even on Mark I: its electromechanical tables took a long time to calculate a value, as they used the full numerical precision of the machine. Programmers Richard Bloch and Grace Hopper soon found it necessary to develop more efficient routines for specific problems. As Mark I only had one sequence mechanism, however, they had no alternative to recording and reusing these routines by hand, as Hopper recalled:

> And if I needed a sine subroutine, angle less than $\pi/4$, I’d whistle at Dick and say, “Can I have your sine subroutine?” and I’d copy it out of his notebook. (Hopper 1981)

It quickly became clear that it would be useful to plan in advance, and to make routines that were likely to be generally useful available for reuse. In 1945, to test the usability of the EDVAC code he had designed, von Neumann wrote a program to merge two sequences of data. After completing the code, he noted the potential generality of the procedure and commented that it could

> be stored permanently outside the machine, and it may be fed into the machine as a “sub routine”, as a part of the instructions of any more extensive problem, which contains one or more [merge] operations. (von Neumann 1945c, 25-6)[10]

Subroutines were extensively discussed by the EDVAC group in the summer of 1945, and in September Eckert and Mauchly provided the following account in a progress report:

> It is by the use of “subsidiary chains” of orders, to be called into use from time to time, as they are needed, by a “higher” set of orders, that a computational routine can be compactly represented. What is more, this corresponds to the way in which mathematical processes are most easily and naturally thought about. The rule for interpolation is not written down anew each time it must be used, but is regarded as a “subsidiary routine” already known to the computer, to be used when needed. (Eckert and Mauchly 1945, 40)

Eckert and Mauchly made here the familiar connection between subroutines and interpolation, and hence the use of tables, but it is striking that the direction of the metaphor is now reversed and the terminology of automatic computing is used to characterize a familiar and long-established mathematical practice.

[10] It is not clear whether the term “subroutine” originated with von Neumann or whether he took it over from the Mark I programmers. Assuming that Hopper in 1981 was not providing a verbatim report of her 1944 conversation with Bloch, von Neumann’s manuscript is the earliest documented usage that I know of, and it is perhaps significant that the term does not appear in (Harvard 1946). In fact, the more general term “routine” seems to appear only once in that volume (on page 98), suggesting that it was not in common use in Harvard.

The idea that subroutines would be recorded in a notebook already seemed outdated, and the benefits of more systematic ways of storing and sharing code were becoming recognized. Herman Goldstine (1945) commented that "[e]vidently one would collect in his library tapes for handling standard types of problems such as integrations and interpolations", and even in Harvard sequence tapes of "general interest" were "preserved in the tape library" (Harvard 1946, 292).

The idea of a subroutine library soon caught on and the developers and users of various machines began to plan standard libraries. As well as convenience, the promise of greater reuse made it economic to analyze the library routines to ensure that they were efficiently coded and would work correctly in a range of contexts. In a January 1947 report on EDVAC programming, Samuel Lubkin (1947, 20, 28) gave an example of a "standard subroutine" to compute square roots "in the form it would take in a library of subroutines", while at around the same time ENIAC operator Jean Bartik was contracted by BRL to run a programming group charged with developing "the technique of programming the production of trigonometric and exponential functions" along with a number of other routines of interest to ballisticians.[11] Some years later, the library concept and techniques for writing and using subroutines were more widely disseminated in the textbook by Wilkes et al. (1951) which made, as its subtitle promised, "special reference to the EDSAC and the use of a library of subroutines".[12]

The metaphor of the "library" is telling. Authors working in libraries consult reference books, but the texts they are writing do not form part of the library. At best, they will be added to the shelves only after being completed, published and found worthy of preservation. Similarly, a subroutine thought to be generally useful might, after extensive checking, be placed in a library, but the main routines written to solve specific problems were treated quite separately and were less likely to be permanently stored. Work practices reinforced the distinction between the two types of code. Wilkes et al. (1951, 43) described how EDSAC subroutines and master routines were punched on separate tapes and only combined at the last minute to form a program tape for an actual computation. The subroutines themselves were punched on coloured tape and stored in a steel cabinet, while the master copies were kept under lock and key. In contrast to these complex and bureaucratized procedures, the master routine tapes could be treated very casually, as the story of Wilkes' Airy program reveals (Campbell-Kelly 1992). At Harvard (1946, 292), there was also a contrast between the care that would go into the preparation of a library tape for Mark I and one intended to be run but once.

Subroutines, then, are a technique with roots in the mathematical practice of table use that allowed programs to be efficiently structured and written. However, while a mathematician carrying out a complex calculation would not normally develop a new interpolation routine, say, programmers did identify new and unanticipated

[11] See (Anonymous 1947) for the complete list of problems assigned to the group. As Bartik (2013, 115-120) described, however, much of the group's effort was diverted to developing EDVAC-style codes in advance of ENIAC's conversion to central control.

[12] Not to be confused with EDVAC, EDSAC was an electronic computer developed in Cambridge by a team led by Maurice Wilkes. It came into operation in 1949.

subroutines while writing new programs. Among the first to notice this were BRL mathematicians Haskell Curry and Willa Wyatt who in 1946 planned an interpolation routine for ENIAC. They divided the program into a number of "stages" and, noting that some stages could be reused to avoid having to recode them, went on to make the methodological recommendation that programmers identify reusable stages by looking for repeated code: "the more frequently recurring elements can be grouped into a stage by themselves" (Curry and Wyatt 1946, 30).

However, other writers did not follow this lead. Subroutines were not explicitly represented in the flow diagram notation, and in the *Planning and Coding* reports Goldstine and von Neumann offered no guidance on how to identify useful new subroutines. Some of the library subroutines described by Wilkes et al. (1951), such as those carrying out integration, made use of "auxiliary subroutines" which defined the function being integrated, but more general uses of user-defined subroutines were not considered.

As a result, perhaps, ad hoc subroutines were rather uncommon in practice. Of the 30 stages in Curry and Wyatt's interpolation program, only four were identified as being reusable. In the Monte Carlo programs run on ENIAC in 1948, there was only one subroutine (to compute a pseudo-random number) in approximately 800 program instructions (Haigh et al. 2016, 183-6). Programming guidelines for the Harvard Mark II even suggested that in general "the method which involves the fewest routines [...] is the logical choice" (Harvard 1949, 266).

The emphatic distinction between master routines and subroutines had another consequence, namely that calling hierarchies were rather flat. Typically, a master routine would call a small number of subroutines, but it was rather rare for one subroutine to call another. The techniques used for subroutine call and return further meant that recursive calls, where a subroutine calls itself, were not possible.

The practices of subroutine use that emerged in the early years of automatic computing, then, reflected the ways in which tables were used in manual calculation. Like a set of tables, a subroutine library is a resource that is available in advance of a computation, and subroutine use was largely restricted to calling routines from a library. Looking up a table is an exceptional task that takes the computer away from the normal process of working through a computation sheet and, similarly, calling a subroutine is an exceptional occurrence. Looking up a table is a self-contained and non-recursive operation: when looking up a value in a table, you rarely have to look up a second table in order to complete the operation. Similarly, complex structures of calling relationships between subroutines appear to be uncommon.

These assumptions were still in evidence ten years later in the first widely-used programming language, Fortran. Like the computer itself, Fortran was intended for mathematical application. The source code was described as "closely resembling the ordinary language of mathematics" and "intended to be capable of expressing any problem of mathematical computation" (IBM 1956, 2). Subroutines were understood by analogy with mathematical functions. A formula containing a function, such as $a - \sin(b - c)$, could be translated directly into Fortran as `A-SINF(B-C)` (IBM 1956, 12). Fourteen functions were provided as "built-in subroutines" of the language, but these were for rather simple operations such as returning the absolute value of a number. Functions that would typically have been tabulated, such as the trigonometric and exponential functions, were not built in and were left for users to define.

However, new subroutines could not be defined in the Fortran language itself, but had to be written in machine code, and then added to the library in rather a complex and labour-intensive process.

> Library subroutines exist on the master FORTRAN tape in relocatable binary form. Placing a new subroutine on that tape involves (1) producing the routine in the form of relocatable binary cards, and (2) transferring these cards on to the master tape by means of a program furnished for that purpose. (IBM 1956, 40)

Only with the arrival of Fortran II in 1958 did the language provide more general support for the definition and use of functions and subroutines.

> The FORTRAN II subprogram facilities are completely general; subroutines can in turn use other subroutines to whatever degree is required. These subroutines may be written in source program language. For example, subprograms may be written in FORTRAN II language such that matrices may be processed as units by a main program. (IBM 1958, 1)

8 Conclusions

This chapter began by considering the view expressed by Davis and Mahoney that since EDVAC the computer has been intrinsically a universal logic machine, and hence that its subsequent application to a host of application areas was, if not always straightforward in practice, at least unproblematic in theory. A consequence of this view is that the computer's origins as a technological innovation to automate specific mathematical processes are reduced to the level of an incidental detail.

In contrast, this chapter has shown that EDVAC, like its predecessors, was planned, promoted, designed and built for very specific mathematical purposes. This perspective dominated much computer development throughout the rest of the 1940s, and I have argued elsewhere (Priestley 2011, 147–153) that the identification of machines based on the EDVAC design with Turing's idea of a universal machine was not widely made until the early 1950s. As Mahoney might have pointed out, the story of the adoption of the computer by non-mathematical communities is often the story of how the mathematical orientation of the early machines was overcome. As Christopher Strachey, one of the first people to write substantial programs for non-mathematical applications, commented:

> the machines have been designed principally to perform mathematical operations. This means that while it is perfectly possible to make them do logic, it is necessarily a rather cumbersome process. (Strachey 1952)

What was invented in the 1940s was not just the automatic computer, however, but modern computing. The machines were conceived as replacements for human computers engaged in mathematical calculation. As Stibitz made clear, this is why they are called computers. The computers' job was to carry out, in ways specified by an explicit plan, a sequence of operations, and the central innovation of modern computing was to automate the task of instruction following. Rather than describing the take-up of a uniquely capable technology, Mahoney's "histories of computing" were to be the stories of how different communities came to reformulate their existing work practices in the form of computer programs.

The task of preparing instructions for the new machines to execute, the activity that we now call programming, naturally became of central importance. Sections 6 and 7 showed how early thinking about programming was profoundly shaped by the mathematical context in which the new computers were built. At the organizational level, existing techniques for managing large-scale calculation were preserved as far as possible. Goldstine and von Neumann's *Planning and Coding* reports dealt largely with mathematical applications and were rooted in a division of labour dating back to the late eighteenth century. Machine-specific techniques were categorized as coding issues, and it was assumed that the overall planning of a computation could proceed along familiar lines. At a more detailed level, the particular ways in which subroutines were used to make programming more efficient reflected aspects of the use of mathematical tables in manual computation. This is not to say, of course, that the use of subroutines was limited to mathematical functions—the EDSAC library also included crucially important input and output subroutines. The point is, rather, that the role of subroutines within programs and the ways in which they were used were constrained by their association with existing practices of using mathematical tables.

These two aspects are characteristic of a general approach to programming that was widely accepted in the late 1940s and early 1950s. Many of the developments of the 1950s, such as the move to automate coding that led to the development of high-level programming languages such as Fortran, were aimed at making technical improvements within this framework but did not break away from the overall model or the mathematically-oriented thinking that underlay it.

The first explicit reflection on and challenge to this approach emerged, perhaps unsurprisingly, in a non-mathematical context. In 1955, Allen Newell and Herbert Simon began to consider the prospects of writing programs to solve what they called "ultracomplicated problems" such as chess playing and theorem proving. They chose the latter as a testbed, and by 1956 had developed the Logic Theorist (LT), a program capable of finding proofs in the propositional calculus. They found existing programming technique inadequate for developing LT, developing instead a notion of "heuristic programming".[13]

Newell and Simon's critique of current approaches to programming focused on precisely the two issues that I have taken as being emblematic of the mathematical approach to programming. They first addressed the belief that computations had to be planned in advance in exhaustive detail.

> But one of the sober facts about current computers is that, for all their power, they must be instructed in minute detail on everything they do. To many, this has seemed to be harsh reality and an irremovable limitation of automatic computing. It seems worthwhile to examine

[13] See (Priestley 2017) for a more detailed account of Newell and Simon's critique and the take-up of their work by the nascent AI community.

> the necessity of the limitation of computers to easily specified tasks. (Newell and Simon 1956, 1)

Secondly, they noted that the design of LT made extensive use of subroutines. Recognizing that "most current computing programs [...] call for the systematic use of a small number of relatively simple subroutines that are only slightly dependent on conditions", they argued for a view of program structure that was quite different from the prevailing view of a program as a sequence of statements. Whereas "[a] FORTRAN source program consists of a sequence of FORTRAN statements" (IBM 1956, 7), Newell and Simon held that:

> a program [...] is a system of subroutines [...] organized in a roughly hierarchical fashion. [...] The number of levels in the main part of LT is about 10, ignoring some of the recursions which sometimes add another four or five levels. (Newell and Shaw 1957, 234-8)

This vision of the use of subroutines is quite different from the prevailing model discussed in Section 7 of this chapter. Rather than corralling subroutines in libraries that enforced limited and rather stereotypical patterns of use, Newell and Simon viewed them as being fundamental programming structures on a par with loops and conditional branching. Their programming work was highly influential in the late 1950s in the emerging field of artificial intelligence (Feigenbaum and Feldman 1963), and it is very striking that in applying automatic computers to this new area of application they rejected two aspects of the traditional approach that directly reflected the specific practices of mathematical computation.

Certain aspects of Newell and Simon's approach can be found in the personal styles of earlier writers. In his proposal for the ACE, Turing gave some examples of the "paper technique of using the machine", culminating in the definition of a routine CALPOL to calculate the value of a polynomial. The program for CALPOL, or "instruction table" in Turing's terminology, made use of eight subsidiary routines, and its code bore out Turing's general comment that:

> The majority of instruction tables will consist almost entirely of the initiation of subsidiary operations and transfers of material. (Turing 1946, 28)

Turing was exceptional among the computer developers of the early 1940s in having no significant experience of large-scale manual computing. The intellectual roots of his famous 1936 paper on computable numbers were in the logical theory of recursive functions, which proceeds by building up complex definitions from simpler ones. Turing adapted this approach for his machine table notation, and the table defining the universal machine is built up largely by combining many simpler tables (Priestley 2011, 77–92). It is precisely this style of thought that is reflected in his practical programming examples such as the table for CALPOL.

Curry and Wyatt's 1946 interpolation program for ENIAC was constructed by combining a large number of small program fragments. Although he spent the war as a BRL mathematician, Curry's background and interests were, like Turing's, in mathematical logic rather than practical computation. In two later reports Curry developed this approach into a general theory of program construction, one that he explicitly opposed to the Goldstine/von Neumann model of subroutines and that bore more than a passing resemblance to his work in combinatory logic (De Mol et al. 2013).

Neither Turing's example nor Curry's theory made an immediate impact, however. Rather than developments in logical theory, it was the stimulus to develop programs for a new class of essentially non-mathematical problems that led, in the mid-1950s, to the establishment of an alternative to the prevailing approach to programming.

Acknowledgements I would like to thank Martin Campbell-Kelly, Tom Haigh, and Sven Ove Hansson for their helpful comments on earlier versions of this chapter.

References

AWB-IUPUI Arthur W. Burks Papers, Institute for American Thought, Indiana University–Purdue University Indianapolis

ETE-UP ENIAC Trial Exhibits, Master Collection, 1964–1973, University of Pennsylvania Archives and Records Center, Philadelphia.

MSOD Moore School of Electrical Engineering, Office of the Director Records, 1931–1948, UPD 8.4, University Archives and Records, University of Pennsylvania, Philadelphia.

Aiken, H. H. (1937). Proposed automatic calculating machine. In Randell (1982), pages 195–201.

Anonymous (1947). Problems 1947-1948. MSOD box 13.

Babbage, C. (1822). *On the Application of Machinery to the Purpose of Calculating and Printing Mathematical Tables*. In M. Campbell-Kelly, editor (1989), *The Works of Charles Babbage*, volume 2, pages 6–14. Pickering.

Bartik, J. J. (2013). *Pioneer Programmer*. Truman State University Press.

Bloch, R. M. (1999). Programming Mark I. In Cohen, I. B. and Welch, G. W., editors, *Makin' Numbers: Howard Aiken and the Computer*. The MIT Press.

Brainerd, J. G. (1943). Letter to Dean Harold Pender, April 26, 1943. MSOD box 51.

Brainerd, J. G. (1944a). Letter to Lt. Col. Paul N. Gillon, September 13, 1944. MSOD box 55a.

Brainerd, J. G. (1944b). Letter to Lt. Thomas H. Bogert, September 13, 1944. AWB-IUPUI.

BRL (1944). Ballistics Research Laboratory Firing Table Reviewing Board, memorandum to Colonel Leslie E. Simon, August 30, 1944. AWB-IUPUI.

Bush, V. (1931). The differential analyzer: A new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447–488.

Campbell-Kelly, M. (1992). The Airy tape: An early chapter in the history of debugging. *IEEE Annals of the History of Computing*, 14(4):16–26.

Curry, H. B. and Wyatt, W. A. (1946). A study of inverse interpolation of the Eniac. BRL Report No. 615. Aberdeen Proving Ground, Maryland. August 19, 1946.

Davis, M. (2000). *The Universal Computer*. W. W. Norton & Company.

Daylight, E. G. (2015). Towards a historical notion of 'Turing—the father of Computer Science'. *History and Philosophy of Logic*, 36(3):205–228.

De Mol, L., Carlé, M., and Bullynck, M. (2013). Haskell before Haskell: an alternative lesson in practical logics of the ENIAC. *Journal of Logic and Computation*, 25(4):1011–1046.

Eckert, J. P. and Mauchly, J. W. (1945). Automatic High Speed Computing: A Progress Report on the EDVAC. Moore School of Electrical Engineering, September 30, 1945.

Eckert, Jr., J. P. (1946). A preview of a digital computing machine. In Campbell-Kelly, M. and Williams, M. R., editors, *The Moore School Lectures: Theory and Techniques for Design of Electronic Digital Computers*, volume 9 of *Charles Babbage Institute Reprint Series for the History of Computing*, pages 109–126. The MIT Press. Lecture delivered July 15, 1946.

Ensmenger, N. (2010). *The Computer Boys Take Over*. MIT Press.

Ensmenger, N. (2016). The multiple meanings of a flowchart. *Information & Culture*, 51(3):321–351.

Feigenbaum, E. A. and Feldman, J. (1963). *Computers and Thought*. McGraw-Hill.

Goldstine, H. H. (1944a). Letter to John von Neumann, October 20, 1944. ETE-UP, Plaintiff's Trial Exhibit Number 2512.

Goldstine, H. H. (1944b). Letter to Lt. Col. P. N. Gillon, August 21, 1944. ETE-UP, Plaintiff's Trial Exhibit Number 2343.

Goldstine, H. H. (1944c). Letter to Lt. Col. P. N. Gillon, September 2, 1944. ETE-UP, Plaintiff's Trial Exhibit Number 2395.

Goldstine, H. H. (1945). Letter to Haskell B. Curry, October 3, 1945. ETE-UP, Plaintiff's Trial Exhibit Number 3556.

Goldstine, H. H. and von Neumann, J. (1947). *Planning and Coding of Problems for an Electronic Computing Instrument*, Volume 1. Institute for Advanced Study, Princeton, April 1, 1947.

Grattan-Guinness, I. (1990). Work for the hairdressers: The production of de Prony's logarithmic and trigonometric tables. *Annals of the History of Computing*, 12(3):177–185.

Grier, D. A. (1998). The Math Tables Project of the Work Projects Administration: The reluctant start of the computing era. *IEEE Annals of the History of Computing*, 20(3):33–50.

Haigh, T. (2014). Actually, Turing did not invent the computer. *Communications of the ACM*, 57(1):36–41.

Haigh, T., Priestley, M., and Rope, C. (2014). Reconsidering the stored-program concept. *IEEE Annals of the History of Computing*, 36(1):4–17.

Haigh, T., Priestley, M., and Rope, C. (2016). *ENIAC in Action: Making and Remaking the Modern Computer*. MIT Press.

Halsey, F. A. (1896). Some special forms of computers. *Transactions of the American Society of Mechanical Engineers*, 18:70–74.

Hartree, D. R. (1949). *Calculating Instruments and Machines*. The University of Illinois Press.

Harvard (1946). *A Manual of Operation for the Automatic Sequence Controlled Calculator*. The Annals of the Computation Laboratory of Harvard University, Volume I, Harvard University Press.

Harvard (1949). *Description of a Relay Calculator*. The Annals of the Computation Laboratory of Harvard University, Volume XXIV, Harvard University Press.

Hering, C. (1891). *Universal Wiring Computer for Determining the Size of Wires for Incandescent Electric Light Leads*. W. J. Johnston, New York.

Hopper, G. M. (1981). Keynote address. In Wexelblat, R. L., editor, *History of Programming Languages*, pages 7–20. Academic Press.

Hurd, C. C. (1985). An note on early Monte Carlo computations and scientific meetings. *Annals of the History of Computing*, pages 141–155.

IBM (1956). *The FORTRAN Automatic Coding System for the IBM 704 EDPM: Programmer's Reference Manual*. Applied Science Division and Programming Research Dept., IBM: New York, NY, October 15, 1956.

IBM (1958). *Reference Manual: FORTRAN II for the IBM 704 Data Processing System*. IBM Corp., New York, NY.

Knuth, D. E. (1973). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.

Lubkin, S. (1947). *Proposed Programmning for the EDVAC*. MSOD, box 8.

Mahoney, M. S. (2005). The histories of computing(s). *Interdisciplinary Science Reviews*, 30(2):119–135.

Mauchly, J. W. (1942). The use of high speed vacuum tubes for calculating. In (Randell 1982, 355–358).

Mindell, D. A. (2002). *Between Human and Machine: Feedback, Control, and Computing before Cybernetics*. John Hopkins University Press.

Moore School (1943). *The ENIAC: a Report Covering Work until December 31, 1943*. University of Pennsylvania, Moore School of Electrical Engineering.

Moore School (1944). *The ENIAC: Progress Report Covering Work from January 1 to June 30, 1944*. University of Pennsylvania, Moore School of Electrical Engineering.

Newell, A. and Shaw, J. C. (1957). Programming the Logic Theory Machine. In *Proceedings of the Western Joint Computer Conference*, pages 230–240.

Newell, A. and Simon, H. A. (1956). *Current Developments in Complex Information Processing*. RAND Corporation Report P-850, May 1, 1956.

Priestley, M. (2011). *A Science of Operations: Machines, Logic and the Invention of Programming*. Springer.

Priestley, M. (2017). AI and the origins of the functional programming language style. *Minds and Machines*, 27(3):449–472.

Randell, B., editor (1982). *The Origins of Digital Computers: Selected Papers (Third Edition)*. Springer-Verlag.

Reed, H. L. (1952). Firing table computations on the ENIAC. In *Proceedings of the 1952 ACM National Meeting (Pittsburgh)*, pages 103–106.

Stibitz, G. R. (1945). *Relay Computers*. Applied Mathematics Panel, National Defense Research Committee. AMP Report 171.1R.

Strachey, C. S. (1952). Logical or non-mathematical programmes. In *Proceedings of the 1952 ACM National Meeting (Toronto)*, pages 46–49.

Turing, A. (1946). Proposed electronic calculator. DSIR 10/385, The National Archives, London, UK.

von Neumann, J. (1944a). Letter to Robert Oppenheimer, August 1, 1944. Los Alamos National Laboratories, LA-UR-12-24686.

von Neumann, J. (1944b). *Proposal and Analysis of a New Numerical Method for the Treatment of Hydrodynamical Shock Problems*. In Taub, A. H., editor (1963), *John von Neumann, Collected Works, Volume VI: Theory of Games, Astrophysics, Hydrodynamics and Meteorology*, pages 361–379. Pergamon Press.

von Neumann, J. (1945a). Memorandum on Mechanical Computing Devices, to Col. L. E. Simon, January 30, 1945. Herman Heine Goldstine Papers, American Philosophical Society, Philadelphia, series 5, box 2.

von Neumann, J. (1945b). *First Draft of a Report on the EDVAC*. Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945.

von Neumann, J. (1945c). Untitled typescript containing discussion of EDVAC program. American Philosophical Society, Mss. 510.78.V89p.

von Neumann, J. (1947). Letter to Robert Richtmyer, March 11, 1947. In Hurd (1985), pages 149–153.

Wilkes, M. V., Wheeler, D. J., and Gill, S. (1951). *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley Press, Inc.

Zuse, K. (1936). Method for automatic execution of calculations with the aid of computers. In Randell (1982), pages 163–170.