The histories of computing(s)

MICHAEL S. MAHONEY

Program in History of Science, Princeton University, Princeton, NJ 08544, USA

The first electronic digital computers were variations on the protean design of a limited Turing machine, which described not a single device but a schema, and which could assume many forms and could develop in many directions. It became what various groups of people made of it. The computer thus has little or no history of its own. Rather, it has *histories* derived from the histories of the groups of practitioners who saw in it, or in some yet to be envisioned form of it, the potential to realise their agendas and aspirations. What kinds of computers we have designed since 1945, and what kinds of programs we have written for them, reflect not so much the nature of the computer as the purposes and aspirations of the communities who guided those designs and wrote those programs. Their work reflects not the history of the computer but the histories of those groups, even as the use of computers in many cases fundamentally redirected the course of those histories. Separating the histories of computing, or perhaps even of computings, shifts attention to the major communities, or bodies of shared disciplinary practices, who embraced the new device and helped to shape it by adapting it to their needs and aspirations.

History of commitments constrains choice.
Narrow incentives and opportunities motivate choice.[1]

The 'digital' in the title of the lecture series from which this paper derives points to the future of the humanities, which for the moment remain still largely 'analogue'. I can't claim strong credentials when it comes to looking toward the future. During my final year at Harvard in 1959/60, I had a job as a computer programmer for a small electronics firm in Boston. It involved writing code for a Datatron 204, soon to become through acquisition the Burroughs 204, a decimally addressed, magnetic drum machine. Programming it meant understanding how it worked, since it was just you and the computer: no operating system, no programming support. Six or seven months of that persuaded me that computers were not very interesting, nor did they seem to me to have much of a future. So I abandoned my thoughts of going into applied mathematics and became a historian instead. With foresight like that, it was probably a good choice.

I'm not sure my foresight improved when I returned as historian to computing in the early 1980s, at least not enough to make the right investments. Yet, I did have enough critical understanding of what was happening to distrust the utopian promises of the time. Henry Adams showed in his *Education* how the past can be a good place to look for the future. History helps to us to know where we might be headed by establishing where we are and how we got here. The future is largely the result of our present momentum. There are surprises, of course, things we don't see coming. Flash Gordon and Buck Rogers were rocketing among the planets in the 1930s without the aid of computers, which we now

DOI 10.1179/030801805X25927

© 2005 Institute of Materials, Minerals and Mining. Published by Maney on behalf of the Institute

know to be indispensable to the enterprise and which made a reality of fiction. But even surprises wind up having a history, first because the new most often comes incrementally embedded in the old, and because in the face of the unprecedented we look for precedents.[2] It is the only way we can understand what is new about something, or the way in which it is new, and adjust our thinking to it.

It is especially important to bear that in mind with respect to computers and computing, because they have always been surrounded by hype (it was – and may still be – the only way to sell them), and hype hides history. From the very beginning people in the field have been engaged in instant historical analysis aimed at declaring a new epoch, a radical disjuncture. Edmund C. Berkeley, the author in 1948 of *Giant Brains, or Machines that Think*, first announced the 'computer revolution' in a book with that title in 1962.[3] Identifying it as an aspect of the 'second industrial revolution', Berkeley combined a gentle introduction to computers with fretting over the social implications of the automation of thought. Since his book, the word 'revolution' has appeared regularly in tandem with 'computer' or 'information', where the latter term has become all but identical with what computers store and manipulate. Were one to take this literature at face value, the job of the historian would seem clear: to chronicle the revolution – or, rather, revolutions, because no area of computing is complete without causing a revolution – and to rank it among the other great revolutions of history: the agricultural revolution, the scientific revolution, the industrial revolution.

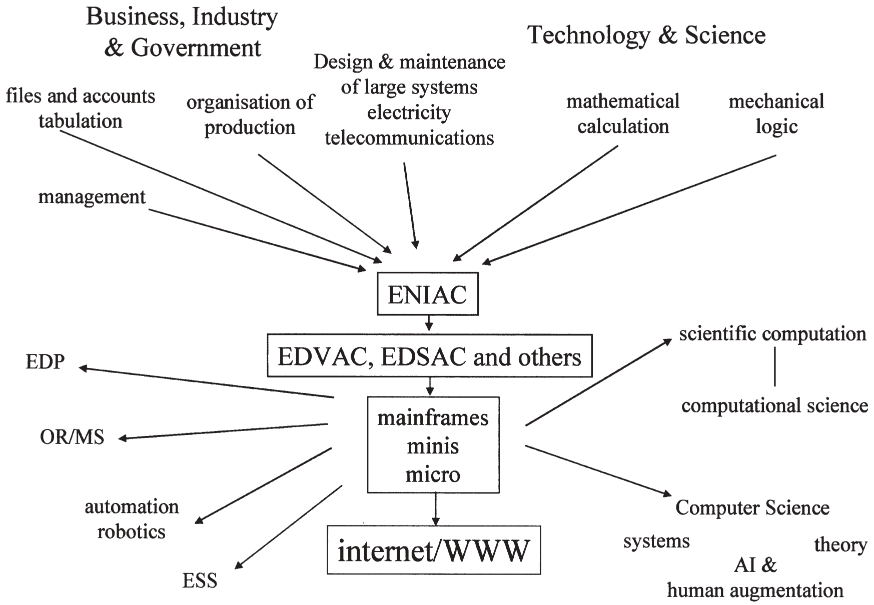
Historians prefer to judge revolutions in retrospect, after the dust has settled. Revolutions aren't what they used to be. The events of the past decades have shown us how hard it is to erase or escape a people's history. As historian of science I'm watching a subject I've taught for more than thirty years, the scientific revolution, declared a non-event.[4] More important, perhaps, most declarations of the 'computer revolution' have rested on future promises rather than on present or past performance. 'See what computers are on the verge of doing. It will be revolutionary!' 'When computers start thinking for themselves, what is going to become of us?' Anyone familiar with the literature of computing over the past forty years knows that the field has been long on promises and short on performance. The literature is filled with revolutionary breakthroughs postponed owing to technical difficulties.

History is the record of our collective experience, our social memory. We turn to it, as we do to our personal experience, consciously when we meet new situations, unconsciously as we live day to day. My research into the formation of two new subjects, theoretical computer science and software engineering, reveals how people engaged in new enterprises bring their histories to the task, often different histories reflecting their different backgrounds and training.[5] The creators and early practitioners of these fields all came from somewhere else. In a recent article, 'Finding a history for software engineering', I have tried to show how efforts to define and articulate a new engineering discipline for software have rested on practitioners' understanding of the history of other fields of engineering.[6] It is incorporated in our institutions, most particularly our schools, where it is taught consciously in history courses, unconsciously in every other part of the curriculum. It is embodied in the artefacts we use, in the customs we follow, in the language we speak. Whether we want to or not, we use history. As in the case of personal memory, the question is whether we use it well. That is a matter both of getting the history right and of getting the right history. As Rob Kling and Walt Scacchi argue in their studies of the social patterns of computing and their impact on systems design, 'history of commitments constrains choices' – even, or especially, revolutionary choices.

DECENTRING THE MACHINE

With some recent exceptions, the history of computing has been centred on the machine, tracing its origins back to the abacus and the first mechanical calculators and then following its evolution through the generations of mainframe, mini, and micro. Alongside this machinic main thread run accounts of scientific calculations; statistics and tabulations; and the growing informational needs of business, industry and government, all converging in the mid 1940s on the electronic digital computer and then spreading out again in new forms shaped by it, in the end to be tied together by the internet (Fig. 1). Once invented, the computer evolves naturally into the PC as its present most visible form, rather than into a variety of coexisting, mutually supportive forms (as if mainframes disappeared with the invention of the minicomputer). Its progress is inevitable and unstoppable, its effects revolutionary.

Chronicling the revolution, that machine-centred history reinforces the hype and with it what one might call the 'impact theory' of the relation of technology and society. There is society strolling along, minding its own business, and, wham!, it gets impacted and is left reeling by a revolutionary technology, which changes everything overnight or in some similarly short time. '[T]he ominous rumble you sense is the future coming at us', wrote one management systems expert in 1953.[7] From that perspective, society breaks up into two classes – those who are on the train and those who are not – and the latter are hopeless (or, as one enthusiast recently put it with a curious nod to Engels, consigned to the dustbin of history). In our field, you're either a digeratus or a dinosaur.


Business, Industry & Government
Technology & Science
files and accounts tabulation
organisation of production
Design & maintenance of large systems electricity telecommunications
mathematical calculation
mechanical logic
management
ENIAC
EDVAC, EDSAC and others
mainframes minis micro
internet/WWW
EDP
OR/MS
automation robotics
ESS
scientific computation
computational science
Computer Science
systems
theory
AI & human augmentation

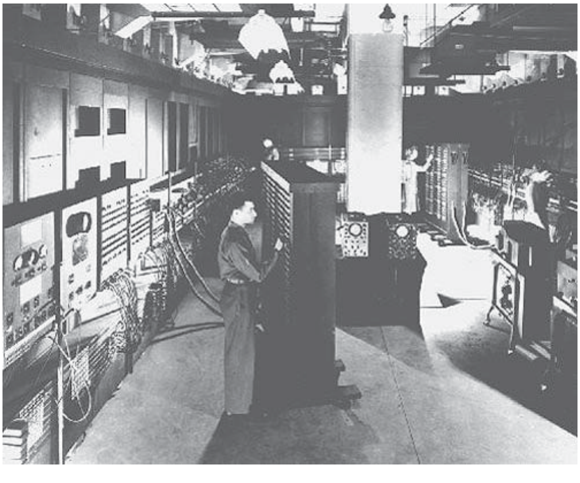
1 The machine centred version of the history of computing

But history of that sort should give us pause on at least two counts. First, computing is a technology (or a constellation of technologies) and, however revolutionary, should have the same sort of richly contextual history that other revolutionary technologies have – such as the steam powered factory and the automobile. The question is not whether new technology involves social change, but how it does. In particular, it is a question of agency. As a form of technological determinism, the impact theory leaves people reacting to technology, rather than actively shaping it. Much of the thoughtful history of technology over the past twenty years has aimed at getting people back into the picture or, to change the metaphor, into the driver's seat. The devices and systems of technology are not natural phenomena but the products of human design, that is, they are the result of matching available means to desired ends at acceptable cost. The available means ultimately do rest on natural laws, which define the possibilities and limits of the technology. But desired ends and acceptable costs are matters of society. Given a set of possibilities, what do we want to do, and what are we willing to pay (in money, time, effort and tradeoffs) to do it?

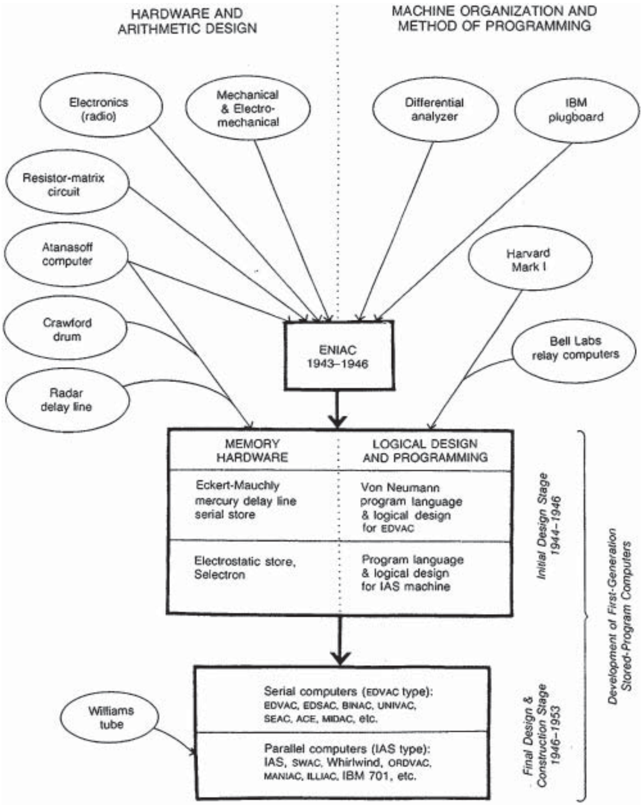
Second, whereas other technologies may be said to have a nature of their own and thus to exercise some agency in their design, the computer has no such nature. Or, rather, its nature is protean; the computer is – or certainly was at the beginning – what we make of it (or now have made of it) through the tasks we set for it and the programs we write for it. Let me take a moment to explain.

THE PROTEAN MACHINE

The new appears to us in contexts defined by the old, by history, or rather histories. Look (Fig. 2) at this picture of the ENIAC, the first electronic digital calculator (not yet a full computer, but its immediate predecessor). It is a new device constructed from existing components, as the scheme by Arthur Burks shows (Fig. 3). Those components embody its history, or rather the history of which it was a product. At this point it was merely doing electronically, albeit for that reason much more quickly, what other devices of the period were doing electromechanically and previous devices had done mechanically, and one can find indications of that throughout its design.



2 The ENIAC in 1946

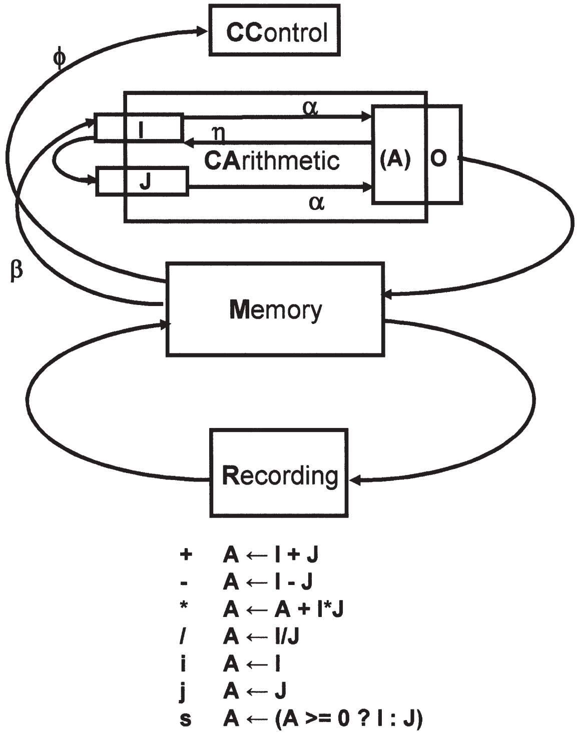

HARDWARE AND ARITHMETIC DESIGN
MACHINE ORGANIZATION AND METHOD OF PROGRAMMING
Electronics (radio)
Mechanical & Electro-mechanical
Differential analyzer
IBM plugboard
Resistor-matrix circuit
Atanasoff computer
Harvard Mark I
Crawford drum
ENIAC 1943–1946
Bell Labs relay computers
Radar delay line
MEMORY HARDWARE
LOGICAL DESIGN AND PROGRAMMING
Eckert-Mauchly mercury delay line serial store
Von Neumann program language & logical design for EDVAC
Electrostatic store, Selectron
Program language & logical design for IAS machine
Initial Design Stage 1944–1946
Development of First-Generation Stored-Program Computers
Williams tube
Serial computers (EDVAC type): EDVAC, EDSAC, BINAC, UNIVAC, SEAC, ACE, MIDAC, etc.
Parallel computers (IAS type): IAS, SWAC, Whirlwind, ORDVAC, MANIAC, ILLIAC, IBM 701, etc.
Final Design & Construction Stage 1946–1953

3 Sources of ENIAC and EDVAC[8]

Only in the next iteration of its design, the EDVAC, could it do things no earlier machine had been able to do, namely make logical decisions based on the calculations it was carrying out and modify its own instructions. The elements of that design have another history, as different from that of ENIAC as the schematic in Fig. 4 is from its circuit diagrams. That is the history of logic machines, reaching back to Leibniz and running through Boole and Turing.[9] The combination of those two histories made the computer in concept a universal Turing machine, limited in practice by its finite speed and capacity. But making it 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).

COMMUNITIES OF COMPUTING

In the early years, then, as the computer moved out of the science and engineering laboratory, it did not bring much history of its own with it. It could not dictate how it was to be used and hence could not have an 'impact' until people figured out what to do with it. Mechanical calculation had a history; symbolic processing did not. Thus the history of computing is the history of what people wanted computers to do and how people designed computers to do it. That may not be one history, or at least it may not be useful



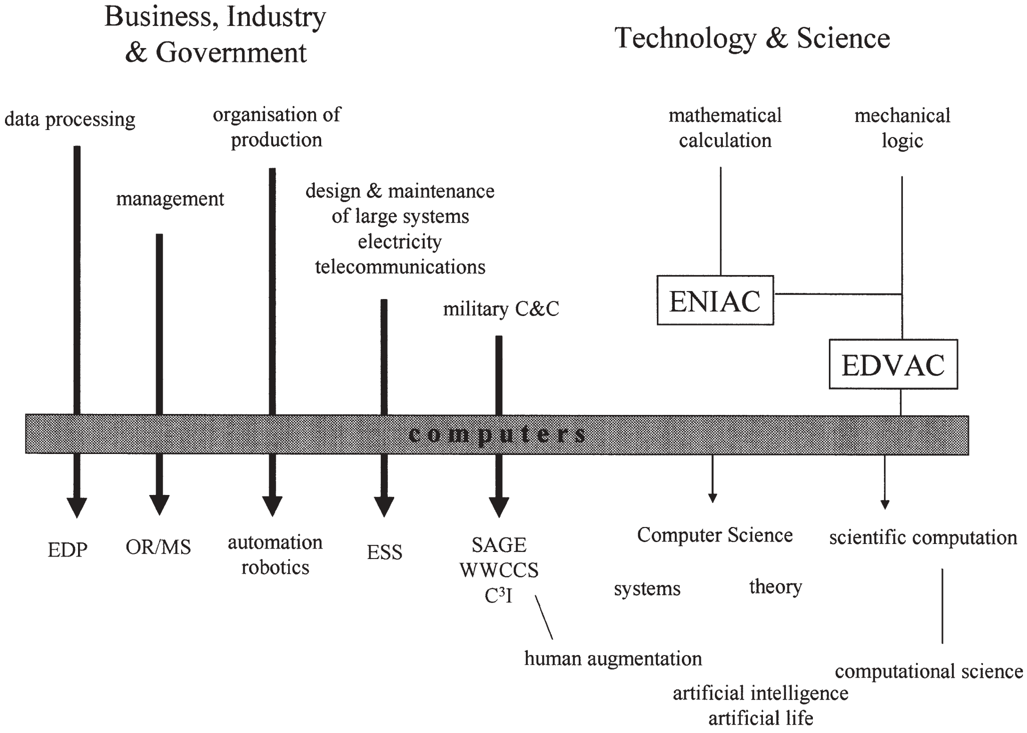
4 Architecture of EDVAC (schematic diagram by the author, based on J. von Neumann: 'First draft of a report on the EDVAC')

to treat it as one. Different groups of people saw different possibilities in computing, and they had different experiences as they sought to realise those possibilities. One may speak of them as 'communities of computing', or perhaps as communities of practitioners that took up the computer, adapting to it while they adapted it to their purposes. The chart in Fig. 5 shows the major groups.

The first, of course, is the scientists and engineers for whose needs the computer had been created. Their practice guided the earliest designs of computers, the series of one-off machines built at government and university sites. They are the community for which the first high level programming language, FORTRAN, was created, a language which continues in use down to the present.

The second group comprises the field of data processing, the first commercial extension of the computer. Recent work by Martin Campbell-Kelly, James Cortada and others (themselves working from an older literature) reveals a rich history going back more than a century. That history explains why IBM took its time in deciding to go into the computer business and why the business took the form it did once the decision had been made.

Tightly connected with data processing is the field of management science, dating back to the work of Frederick W. Taylor and others at the turn of the last century, and beyond that perhaps to Charles Babbage's *On the Economy of Machinery and Manufactures* (1832). Operations research (or in England, operational research) brought a new mathematical dimension to the field as practitioners sought to apply methods of tactical evaluation to business and industry.


Business, Industry & Government
Technology & Science
data processing
management
organisation of production
design & maintenance of large systems
electricity
telecommunications
military C&C
mathematical calculation
mechanical logic
ENIAC
EDVAC
computers
EDP
OR/MS
automation
robotics
ESS
SAGE
WWCCS
C³I
human augmentation
Computer Science
systems
theory
artificial intelligence
artificial life
scientific computation
computational science

5 The communities of computing

Computers also fit into the agenda of industrial engineering in the area of automation and control of production flow, which was given its twentieth century shape by Henry Ford but which can arguably be traced back to Matthew Boulton and James Watt.[10] Indeed, as just noted, it is here, rather than to the Analytical Engine, that one should be looking to place Charles Babbage into the genealogy of computing.

Having already created some of the first electromechanical computing devices to address problems of network analysis and switching systems, the communications industry quickly adopted the electronic computer and began the process of building it into our environment.

Automation of military command and control systems (SAGE and WWMCCS)[11] reflects the history of modern warfare, which reaches back several centuries. In light of recent literature on the subject, we need a fuller and more subtle understanding than we have of the history of the community of practitioners in the military, to compensate for the visions of the future designed for it by researchers outside it. An important spinoff of this line of research was the field of human–computer interaction, aimed at the augmentation of human skills by the computer. As David Mindell has shown recently in *Between Humans and Machines*, that field too had a history of its own before the computer.

New fields emerged alongside those that predated the computer. Although there was at first some disagreement about whether computers were sufficiently different from early calculating machines to constitute a subject of study in themselves, the work of John von Neumann and Alan Turing pointed to the theoretical potential of the device. Moreover, the need to develop systems to both make the computer easier to use and to keep it running efficiently also opened a range of theoretical and technical issues that prompted the emergence of the new subject of computer science or informatics, as it is known in the non-anglophone world.

Until recently the history of computing in these fields has been written in terms of the machine and its impact (revolutionary, of course) on them. The emphasis has lain on what the computer could do rather than on how the computer was made to do it. As I said, in many cases their existence as fields of computing has led to their being written into the history leading up to the computer as if the purposes to which they put the computer had somehow previously constituted a demand for which the computer had been created. Within the last few years, however, that view has begun to shift. In *Information Technology as Business History*, James Cortada justifies a chapter on the '100-year history of mechanical devices used in modern times to do data processing' by admonishing:

> Why do we care? Many of the ways you and I receive computers, buy them, and use this technology were worked out in companies that existed ten, thirty, even fifty years before the first commercial computer was available. Did you know that National Cash Register (NCR) and Burroughs – in time both peddlers of computers – had been selling information-hand[l]ing machines over 100 years ago – or, that International Business Machines Corporation (IBM) has been around since before World War I? Historical context is so important; computers have a lot of it, and its patterns of behavior are described in this chapter.[12]

Tom Haigh, whose work on the history of information systems is beginning to appear, reinforces the point:

> Work by historians such as Martin Campbell-Kelly, JoAnne Yates, and William Aspray has consistently shown that the computer industry was, more than anything else, a continuation of the pre-1945 office equipment industry and in particular of the punched card machine industry. Their careful exploration of computer technology and the dynamics of the computer hardware industry leave little doubt that IBM's eventual dominance of the computer industry owes as much to the events of the 1930s as to those of the 1960s. This is in itself a major departure from the perception, common during the 1950s and common today, that each new generation of computer equipment is a revolutionary technology without historical roots, a breakthrough plucked fully formed from the forehead of (to mix a metaphor) Prometheus.

When looking at the introduction of computing into the business world, Haigh insists that we must break away from a focus on computers.

> The use of computer technology in a particular social space (such as the laboratory, office, or factory) cannot be addressed without also studying the earlier history of this setting, the people in it, and the objectives to which the machine is put. So, while coherent one-volume histories of the computer hardware industry and its technologies can be written, it seems unlikely that we can produce a single coherent narrative about the use of computers or of associated tasks such as analysis, programming, or operation.[13]

Approaching data processing from this wider perspective reveals a history quite different from the current history of computing.

Taking a similarly inspired approach in *The Government Machine*, Jon Agar gives an idea of how different that history looks for the development of computing in Britain. Agar looks back to the eighteenth century, when political thinkers began to speak of 'the machinery of government', and follows the development of that metaphor, as it led to the ever widening collection of data about the population and the introduction of machines of various sorts to record and manipulate the data. Sceptical of the notion of a computer based 'information revolution' and the historical discontinuity it implies, Agar focuses rather on 'the humans who promoted machines' and on the technocratic 'vision of government' that conditioned the adoption of office technologies, including the computer. Indeed, he maintains, '[t]he Civil Service, as a general-purpose universal machine, framed the language of what a computer was and could do.'[14]

As Cortada, Haigh and Agar suggest, the histories and continuing experience of the various communities show that they wanted and expected different things from the computer. They encountered different problems and levels of difficulty in fitting their practice to it. As a result, they created different computers or (if we may make the singular plural) computings. To do so, they had to determine which aspects of their practice were suitable for automation, they had to build computational models of those aspects, and they had to write the programs that implemented those models. None of this was straightforward, except where it was trivial. From the case studies we do have, we can guess that it was a matter of negotiation, both among practitioners about the nature of their practice and between practitioners and the realities of the current technology, often in the shape of non-practitioner technicians.

Until recently, histories of computing have largely ignored that process of negotiation, and that should concern more than historians. From the early 1950s down to the present, various communities of computing have translated large portions of our world – our experience of it and our interaction with it – into computational models to be enacted on computers, not only the computers that we encounter directly but also the computers that we have embedded in the objects around us to make them 'intelligent' or even, as Yorick Wilks would have it, 'companionable'. They have increasingly made computers the medium of our working (in the broadest sense of the term) in the world. In doing so, they have (re)shaped not only their own practice, but also computers and their adaptation by others. So far, we know little of the process by which they have done it. We have the story of where the physical devices came from, how they have taken their current form, and what differences they have made. But we remain largely ignorant about the origins and development of the dynamic processes running on those devices, the processes that determine what we do with computers and how we think about what we do. The histories of computing will involve many aspects, but primarily they will be histories of software.

WORLDS OF SOFTWARE

Historians of computing have only begun to tackle the history of software. We've stuck pretty close to the machine. We know a great deal about the history of programming languages and considerably less about the history of operating systems, databases and other varieties of systems software. We have scarcely scratched the surface of applications software, the software that actually gets things done outside the world of the computer itself.

A recent conference at the Nixdorf Museum in Paderborn, Germany, attempted to map the history of software, considering it as science, engineering, labour process, reliable artefact and industry, with a look at the question of how one exhibits it in a museum.[15] The focus lay on software and its production as general phenomena. What the conference missed was software as model, software as experience, software as medium of thought and action, software as environment within which people work and live. It did not consider the question of how we have put the world into computers.

That process has not been easy or straightforward. If it appears so, it is because so far we have concentrated on the success stories and told them in a way that masks the compromises between what was intended and what could be realised. Programming is where aspiration meets reality. The enduring experience of the communities of computing has been the huge gap between what we can imagine computers doing and what we can actually make them do. There have been (and continue to be) some massive failures in software development, which have cost money, time, property and even lives. Indeed, since the late 1960s people in the field have spoken of a 'software crisis'. Yet, except for Frederick P. Brooks's famous *Mythical Man-Month*, we do not have substantive accounts of those failures, not even in the software engineering classroom, where we might expect to find them. As software engineer/historian James Tomayko points out, other branches of engineering learn from their mistakes.[16] As Henry Petroski puts it in the title of one of his books on the role of failure in design, 'to engineer is human'.[17] Software will look more human when we take seriously the difficulties of designing and building it.

Let me say a bit more about the 'software crisis'. It emerged in the late 1960s as an increasing number of large projects experienced large cost overruns, missed deadlines and failures to meet specifications. Some had to be cancelled altogether. One response was calls for a discipline of 'software engineering' that would, in the words of a deliberately provocative definition, base 'software manufacture . . . on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering'.[18] (That is, of course, a historical programme.) At first, practitioners sought a solution in the computer and looked to the improvement of their tools and at more effective project management. A great deal of effort went into developing high level programming languages and diagnostic compilers for them. The languages were specifically designed to foster good programming practices (read, proper ways to think about programming). Similarly, practitioners sought to bring the experience of industrial engineering to bear on software production, with an eye toward automating it in the form of a 'software factory', a programming environment that would leave the programmer little choice but to do it right.

The tools clearly got better. Once a project gets down to the actual programming, things go relatively smoothly. But, as Brooks has since pointed out, that is not where the real problems have lain, or rather problems at that level were only 'accidental'.[19] Almost from the start studies showed that the bulk of the errors occurred at the beginning of projects, before programming ever began (or should have begun). The errors were rooted in failures to understand what was required, to specify completely and consistently how the system was supposed to behave, to anticipate what could go wrong and how to respond, and so on. As many as two-thirds of the errors uncovered during testing could be traced back to inadequate *design*; the longer they remained undetected, the more costly and difficult they were to correct.

Design is not primarily about computing as commonly understood, that is, about computers and programming. It is about modelling the world in the computer, about computational modelling, about translating a portion of the world into terms a computer can 'understand'. Here it may help to go back to the protean scheme to recall what computers do. They take sequences, or strings, of symbols and transform them into other strings. The symbols and the strings may have several levels of structure, from bits to bytes to groups of bytes to groups of groups of bytes, and one may think of the transformations as acting on particular levels. But in the end, computation is about rewriting strings of symbols.

The transformations themselves are strictly syntactical, or structural. They may have a semantics in the sense that certain symbols or sequences of symbols are transformed in certain ways, but even that semantics is syntactically defined. Any meaning the symbols may have is acquired and expressed at the interface between a computation and the world in which it is embedded. The symbols and their combinations express representations of the world, which have meaning to us, not to the computer. It is a matter of representations in and representations out. What characterises the representations is that they are operative. We can manipulate them, and they in turn can trigger actions in the world. What we can make computers do depends on how we can represent in the symbols of computation portions of the world of interest to us and how we can translate the resulting transformed representation into desired actions. We represent in a variety of forms a Boeing 777 – its shape, structure, flight dynamics, controls. Our representations not only direct the design of the aircraft and the machining and assembly of its components, but they then interactively direct the control surfaces of the aircraft in flight. That is what I mean by 'operative representation'.

So putting a portion of the world into the computer means designing an operative representation of it that captures what we take to be its essential features. That has proved, as I say, no easy task; on the contrary it has proved difficult, frustrating and in some cases disastrous. It has most recently moved to a high priority problem at the US National Science Foundation, which has announced a programme aimed at exploring the 'science of design'.[20] Where that will go is anyone's guess, and I'm a poor prognosticator. What is clear is that historians of computing have inherited the problems to which it is addressed. If we want critical understanding of how various communities of computing have put their portion of the world into software, we must uncover the operative representations they have designed and constructed, and that may prove almost as difficult a task.

READING MACHINES, REAL AND VIRTUAL

What makes it difficult is precisely that the representations are operative. Ultimately it is their behaviour rather than their structure (or the fit between structure and behaviour) that interests us. We do not interact with computers by reading programs; we interact with programs running on computers. The primary source for the historian of software is the dynamic process, and, where it is still available, it requires special techniques of analysis. Programs and processes are artefacts, and we must learn to read them as such.

A brief digression is necessary here. My return to computing occurred when I started teaching the history of technology. Up to that point I had taught history of science from Antiquity through the Scientific Revolution, and I based my courses almost entirely on primary sources. My students read Plato, Aristotle, Aquinas, Copernicus, Galileo and so on. So I looked for primary sources on mills, steam engines, automobiles and computers, and had great difficulty in finding what I wanted. It soon dawned on me that I was looking for the wrong thing in the wrong place. The sources I needed were not texts about these machines, but the machines themselves, to be found not in a library but in a museum. Technology is not a literate enterprise, but a visual, tactile one. Its practitioners think not with words but, as Derek de Solla Price so deftly phrased it, 'with their fingertips'. Henry Ford put it another way:

> There is an immense amount to be learned simply by tinkering with things. It is not possible to learn from books how everything is made – and a real mechanic ought to know how nearly everything is made. Machines are to a mechanic what books are to a writer. He gets ideas from them, and if he has any brains he will apply those ideas.[21]

What holds for the practitioners is true also of those who use technologies, or as Langdon Winner insists, live them.[22] We do not read about them, we act with, in and through them. Conversely, their design assumes that we know certain things or can learn them. They are artefacts of our *culture*, embodying both its explicit and its tacit knowledge.

In the realm of computing, the notion of reading artefacts transfers more or less readily to the machine itself. In *The Soul of a New Machine*, Tracy Kidder relates the story of computer designer Tom West sneaking a peek at a competitor's design:

> Looking into the VAX, West had imagined he saw a diagram of DEC's corporate organization. He felt that the VAX was too complicated. He did not like, for instance, the system by which various parts of the machine communicated with each other; for his taste, there was too much protocol involved. He decided that VAX embodied flaws in DEC's corporate organization. The machine expressed that phenomenally successful company's cautious, bureaucratic style. Was this true? West said it didn't matter, it was a useful theory.[23]

But reading the software is even trickier, because we can't 'pull the boards'. We must learn to interrogate the artefact in action, and here we need help from sociologists, anthropologists and the HCI community, whose studies of current users may suggest interpretive approaches to the past.

Here historians of software face a problem caused by the rapid rate of obsolescence that has characterised computing from the outset. For the early period that particularly interests us, we cannot read the dynamic artefacts, because we no longer have the platforms, the machines and operating systems, on which the software ran. In some cases the disappearance of the platform has meant the loss of the software as well.[24] Given present trends, there is no reason to think that will not be the case with more recent software unless we embark on a systematic programme to archive software and hardware in ways that allow retrieval of the dynamic artefact. It is a bit ironic that in an age that seems overwhelmed by information, the history of the technology designed to manage it will itself be hampered by lack of information, as a crucial body of primary sources is lost to us through the disappearance of the machines on which they were enacted.

In the absence of the running process, the next best thing is the program text, the source code. Again, as historians we inherit the unsolved problems of the subject we are studying. As Christopher Langton, lead proponent of Artificial Life, has put it:

> We need to separate the notion of a formal specification of a machine – that is, a specification of the *logical* structure of the machine – from the notion of a formal specification of a machine's behaviour – that is, a specification of the sequence of transitions that the machine will undergo. In general, we cannot derive behaviours from structure, nor can we derive structure from behaviours.[25]

Despite impressive work in the mathematical theory of program syntax and semantics, we have no means of deriving the dynamic process from the static program, in the sense of being able to determine from the latter the state of the former at some particular point in the computation. Perhaps that should not surprise us. If we did, we would not need the computer.

Nonetheless, if we have the program, we can try to reconstruct from it what the process looked like, and we can learn to analyse the text to discover the structures and operations of the computational model and, through them, to gain some sense of the understandings and intentions of its designers. Alan Kay, the creator of Smalltalk, offers an example through his own discovery of how the language Simula worked. It was his first day as a graduate student at the University of Utah:

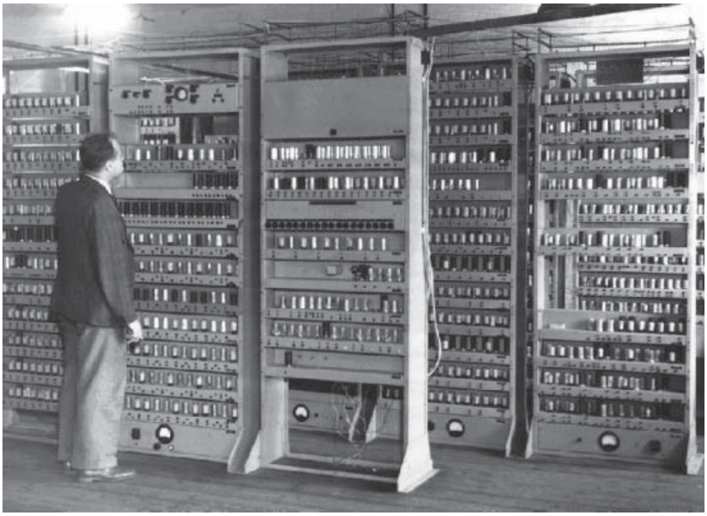
> Head whirling, I found my desk. On it was a pile of tapes and listings, and a note: 'This is the Algol for the 1108. It doesn't work. Please make it work.' The latest graduate student gets the latest dirty task.
>
> The documentation was incomprehensible. Supposedly, this was the Case Western Reserve 1107 Algol – but it had been doctored to make a language called Simula; the documentation read like Norwegian transliterated into English, which in fact was what it was. There were uses of words like *activity* and *process* that didn't seem to coincide with normal English usage.
>
> Finally. another graduate student and I unrolled the listing 80 feet down the hall and crawled over it yelling discoveries to each other. The weirdest part was the storage allocator, which did not obey a stack discipline as was usual for Algol. A few days later, that provided the clue. What Simula was allocating were structures very much like instances of Sketchpad.[26]

Kay's experience suggests what faces the historian, and there is again some irony in it. It has been the common lament of management that programs are built by tinkering and that little of their design gets captured in written form, at least in a written form that would make it possible to determine how they work or why they work as they do rather than in other readily imaginable ways. Moreover, what programs do and what the documentation says they do are not always the same thing. Here, in a very real sense, the historian inherits the problems of software maintenance: the farther the program lies from its creators, the more difficult it is to discern its architecture and the design decisions that inform it.

But at least we have the program texts. Or do we? It is not clear, in part perhaps because for many of the communities of computing outside computer science we have not yet begun to look. When government and industry went looking in the late 1990s in anticipation of Y2K, the results were dismaying. But that is another, different problem.

HISTORY, COMPUTING AND THE HUMANITIES

What does this mean for digital scholarship and our digital future? Let me go back to the beginning. Look at the picture in Fig. 6 of the EDSAC at Cambridge, the first operational computer, and then look at a laptop. A room full of electronic equipment has become a typewriter connected to a TV screen by a black box, with sockets in the back for attachments. What separates the two devices is not evolution but social construction, a lot of it. The difference is not the result so much of working out principles as of pursuing the possibilities of practice.



6 The EDSAC, built by Maurice Wilkes and his team at Cambridge University, 1949

As striking as is the contrast in external form of the two computers, what really separates the EDSAC from my laptop is something on the order of three gigabytes of stored programs, a large number of which simply establish a platform for the programs that I use to do what I want. For the lecture on which this paper is based, I ran a single application on my laptop to project images onto a screen. Enabling that application were some two dozen processes, each in itself a program of some complexity. All these programs reflect the histories of the communities from which they come. The operating system, for example, embodies the history of corporate organisations designed to distribute responsibilities and authority in a hierarchical structure. The graphical user interface, known for its main features as 'WIMP' (windows, icons, mouse, pull-down menus), emerged from the human augmentation community, with its roots in behaviourist perceptual psychology and military command and control systems. Microsoft PowerPoint reflects the adaptation of computer assisted design to the needs of management systems and the corporate boardroom. The communications community provides the networking. And so on. To 'use' a 'personal' computer today is, despite its much hyped origins in the counterculture, to work in a variety of environments created by a host of anonymous people who have made decisions about the tasks to be done and the ways they should be done. As most of us use a computer, it is no more personal than a restaurant: you can have anything you want on the menu, cooked the way the kitchen has prepared it.

Is that bad? No, it is the nature of a technological system. It is the price of computing power. I've worked on a computer without an operating system or a library of programs. The experience drove me from the field. I'm as happy to use the current computing technology as I am to fly on a 777, drive a maintenance-free automobile, or wear no-iron shirts. It is not bad, but it does have implications for the critical, reflective use of computers, especially in the humanities. It means that the computer as tool and medium is not neutral, but rather informs (or, as Bolter and Grusin put it, re-mediates) the work that one does with it, if only by setting possibilities and limits on what can be done (or even thought). It calls for critical awareness. Like the historians of computing, digital scholars must learn to read software to elicit the history and practice that it embodies.

We must be aware because, by and large, we in the humanities have had to borrow our computing from other communities. In the beginning, we simply couldn't afford it on our own. Since then, we have not formed a significant community giving shape to computers by creating our own computing – except perhaps most recently in the design of markup languages and the semantic web. My experience at Princeton is arguably typical. Until the mid 1980s, the scientists and engineers owned the mainframe and the minicomputers, allowing the humanists a bit of time on the side. The personal computer came onto the Princeton campus in the mid 1980s as part of an IBM sponsored project to make the PC an educational resource. IBM supplied machines and basic software, the universities were supposed to generate applications for teaching and research. But, unless the project aimed at a single target, such as a graphically displayed database of US census data down to the county level, not much happened. One reason was that we were trying to develop educational applications using software designed for business. The same was true in the primary and secondary schools, indeed to a greater degree, since they lacked any resources for independent development. (I happened to be a member of our local school board at the time, so could observe developments at both levels.) Either one used the educational software packages of the day (the less said about them, the better), or one tried to adapt to the classroom software originally developed for the business office. Tools embody history. We were trying to work with other people's histories, unaware of what that meant.

Our own history poses a difficult challenge to the computing community. At a workshop for a large project, the National Initiative to Network the Cultural Heritage (NINCH), jointly sponsored by the American Council of Learned Societies and the National Academy of Engineering, humanists and computer scientists gathered to talk about what humanists do, how they do it, what they'd like to be able to do and how they'd like to be able to do it, and how those wishes might constitute research projects for the computer scientists. I was moderating the discussion among the historians and sensed an uncharacteristic humility among my colleagues, who seemed worried about finding something difficult enough to get the attention of the computer scientists. I tried to reassure them that should not be a concern. Our problems are far too difficult for the computer scientists. We need to find something simple enough for them, yet interesting enough for us. Our main tasks involve the sophisticated use of a natural language, or even of several natural languages; the discerning of subtle patterns of shape, colour, texture and sound. Ours is an enterprise of metaphor, analogy, allusion, ambiguity, etc. These are not things that have so far lent themselves to computational modelling. The humanities involve those aspects of human thinking and cognition that have so far confounded artificial intelligence.

The future of digital scholarship depends on whether we can now design computational models of the aspects of the world that most interest us. That, in turn, calls for reflection on our current practice. In seeking to do things in new ways with a computer, it is useful to clarify how we do them now and how we came to do them that way and not otherwise. Indeed, as Willard McCarty has pointed out, grappling with new technology reminds us that we have been using technology all along: pencil, pen and paper are technologies, and all forms of calculation involve instruments.[27] In *Remediation*, a study of the digital medium in the arts and literature, Jay Bolter and Richard Grusin play on the ambiguity of the title, arguing that every re-mediation, every transfer of an activity from one medium to another, rests on a claim of remediation, of making things better.[28] What are we seeking to remedy in the re-mediation of scholarship in the humanities? What do we want to do that, to borrow from Andrew Marvell, we have lacked 'world enough and time' to do? The computer, a coy mistress indeed, offers to supply that world and time, provided that we know how to do what we want and can explain it in terms the computer can understand. Humans have to think hard about both questions. Compared to them, the programming will be easy.

NOTES

1. R. Kling and W. Scacchi: 'The web of computing', *Advances in Computers*, 1982, **21**, 1–90, 39, 56–60.
2. Marx made the point in *Capital*, chapter 15 ('Machinery and large scale industry'), footnote 18: 'To what extent the old forms of the instruments of production influence their new forms at the beginning is shown among other things, by the superficial comparison of the present power-loom with the old one, of the modern blowing apparatus of a blast-furnace with the first inefficient mechanical reproduction of the ordinary bellows, and perhaps more strikingly than in any other way by the fact that, before the invention of the present locomotive, an attempt was made to construct a locomotive with two feet, which it raised from the ground alternately, like a horse. It is only after a considerable development of the science of mechanics, and an accumulation of practical experience, that the form of a machine becomes settled entirely in accordance with mechanical principles, and emancipated from the traditional form of the tool from which it has emerged.' (K. Marx: *Capital*, (trans. Ben Fowkes), Vol. I, 505; 1977, New York, NY, Vintage.)
3. E. C. Berkeley, The Computer Revolution (Garden City: Doubleday, 1962).
4. Thus the opening words of Steven Shapin's *The Scientific Revolution* (1996, Chicago, IL, University of Chicago Press): 'There was no such thing as the Scientific Revolution, and this is a book about it.'
5. See, for example, M. S. Mahoney: 'Computer science: the search for a mathematical theory', in *Science in the Twentieth Century*, (ed. John Krige and Dominique Pestre), chapter 31; 1997, Amsterdam, Harwood Academic.
6. M. S. Mahoney: 'Finding a history for software engineering', *IEEE Annals of the History of Computing*, 2004, **26**, (1), 8–19.
7. Quoted in T. Haigh: 'The chromium-plated tabulator: institutionalizing an electronic revolution, 1954–1958', *IEEE Annals of the History of Computing*, 2001, **23**, (4), 77. Despite the title, Haigh emphasises the slow, evolutionary process of the transition from punched-card machinery to computer systems.
8. As set out in A. W. Burks and A. R. Burks: 'The ENIAC', *Annals of the History of Computing*, 1981, **3**, 318.
9. For that history, see inter alia M. Davis: *The Universal Computer: The Road from Leibniz to Turing*; 2000, New York, NY/London, W. W. Norton, and S. Krämer: *Symbolische Maschinen: Die Idee der Formalisierung in geschichtlichem Abriss*; 1988, Darmstadt, Wissenschaftliche Buchgesellschaft.
10. See, for example, J. Tann: *The Development of the Factory*; 1970, London, Cornmarket Press, which is based on the papers of the Boulton and Watt company.
11. SAGE stands for Semi-Automatic Ground Environment, WWMCCS for World-Wide Military Command and Control System.
12. J. W. Cortada: *Information Technology as Business History: Issues in the History and Management of Computers*, ix; 1996, Westport, CT, Greenwood Press.
13. T. Haigh: 'The chromium-plated tabulator', pp. 75, 94 (see Note 7).
14. J. Agar: *The Goverment Machine: A Revolutionary History of the Computer*, 3; 2004, Cambridge, MA, MIT Press.
15. U. Hashagen, R. Keil-Slawik and A. Norberg (ed.): *History of Computing: Software Issues*; 2002, Berlin/New York, NY, Springer-Verlag.
16. J. E. Tomayko: 'Software as engineering', in *History of Computing*, pp. 65–76 (see Note 15).
17. H. Petroski: *To Engineer is Human: The Role of Failure in Successful Design*; 1985, New York, NY, St. Martin's Press.
18. P. Naur and B. Randell (ed.): *Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, 13; 1969, Brussels, NATO Scientific Affairs Division. The report was republished, together with the report on the second conference in Rome the following year, in

P. Naur, B. Randell and J. N. Buxton (ed.): *Software Engineering: Concepts and Techniques. Proceedings of the NATO Conferences*; 1976 New York, NY, Petrocelli. Randell has made both reports available for download in pdf format at homepages.cs.ncl.ac.uk/brian.randell/NATO.

19. F. P. Brooks: 'No silver bullet – essence and accidents of software engineering', *Information Processing 86*, (ed. H. J. Kugler), 1069–1076; 1986, Amsterdam, Elsevier Science. This essay has been reprinted in *Computer*, 1987, **20**, (4), 10–19 and in the anniversary edition of *The Mythical Man-Month: Essays on Software Engineering*, chapter 16; 1995, Reading, MA, Addison-Wesley, chapter 17 of which, '"No silver bullet" refired', is a response to critics of the original article and a review of the silver bullets that have missed the mark over the intervening decade.
20. See www.nsf.gov/funding/pgm_summ.jsp?pims_id=12766&org=CNS.
21. H. Ford: *My Life and Work*, 23–24; 1992, Garden City, NY, Doubleday. For a reading of Ford's Model T, see my own web document, 'Reading a machine', www.princeton.edu/~hos/h398/readmach/modelt.html.
22. L. Winner: *Autonomous Technology: Technics-out-of-Control as a Theme in Political Thought*, 202; 1977, Cambridge, MA, MIT Press ('We do not use technologies so much as live them.') In a talk some years ago, Ruth Schwarz Cowan, author of *More Work for Mother*, related how living for a week in Bethpage Village, a historical reconstruction of an eighteenth century homestead on Long Island, reshaped her relations with her daughters through the cooperative patterns of work required by the household technology of the time.
23. T. Kidder: *The Soul of a New Machine*, 26; 1981, Boston, MA, Little, Brown.
24. This situation has implications reaching beyond the history of computing. US government records of the 1950s now sit on magnetic tapes that cannot be read because neither the software nor the hardware with which they were created has survived.
25. C. G. Langton: 'Artificial life', in *The Philosophy of Artificial Life*, (ed. M. A. Boden), 47; 1996, Oxford, Oxford University Press. (Langton's essay was first published in 1989.)
26. A. C. Kay: 'The early history of Smalltalk', in *History of Programming Languages – II*, (ed. T. J. Bergin, Jr and R. G. Gibson, Jr) 516; 1996, New York, NY, ACM Press.
27. W. McCarty: 'As it almost was: historiography of recent things', *Literary and Linguistic Computing*, 2004, **19**, (2), 161–180.
28. J. D. Bolter and R. Grusin: *Remediation: Understanding New Media*; 1999, Cambridge, MA, MIT Press.

Michael S. Mahoney (mike@princeton.edu) is Professor of History in the Program in History of Science at Princeton University, where he earned his PhD in history in 1967. His research has focused on the development of the mathematical sciences from Antiquity to 1700 and on the recent history of computing. He is the author of *The Mathematical Career of Pierre de Fermat, 1601–1665*, a series of monographs on the mathematics of René Descartes, Isaac Barrow, Christiaan Huygens and Isaac Newton, and studies on the development of computer science and software engineering.