DOI:10.1145/2500131

Thomas Haigh

Historical Reflections

Software and Souls; Programs and Packages

How can historians tell stories about software without focusing solely on the code itself?

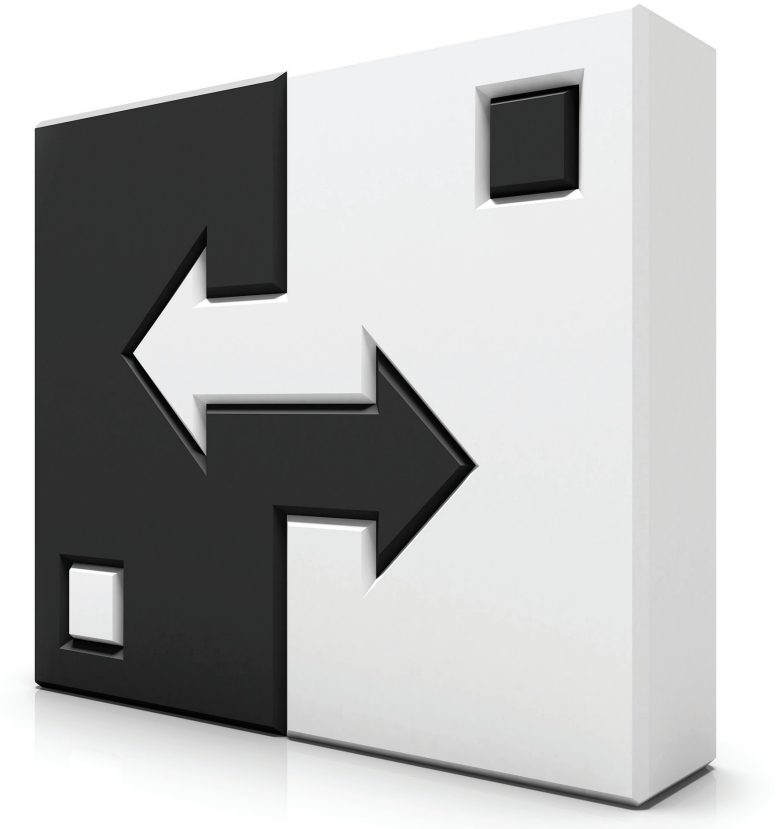

IMAGE FROM SHUTTERSTOCK.COM

Despite his central place in the development of Western philosophical thought, René Descartes is only occasionally cited in computing literature. However, the way we think about hardware and software tends to follow the template established by centuries of Dualist philosophy and thousands of years of human spiritual belief. Hardware is the body. Encumbered by the material world it is constrained by physical laws and will eventually succumb to the ravages of time. Software is the soul. The essence of code is its immateriality. An invisible spark of life, it controls the operation of the machine and can transmigrate from one host to another. It is bound not by the laws of this world but by those of another. The distinction between hardware and software partitions the careers, journals, conferences, interest groups, and businesses of computing into separate camps. In recent years it has also shaped the work of historians of computing, as software history has become an increasingly popular area of research.

This column explores some of the fundamental challenges software poses to historians and draws upon my own historical research in the area to argue that software never really exists in a pure, non-material form. Instead, software has always been treated historically as a "package" including more than just computer code—though what has been meant by this has changed significantly over time.

However, as you are reading *Communications* you probably know a lot about the insides of computers and may very well be an academic computer scientist. So let me admit two objections to my opening generalization before I move on with the historical reflections. First, the lines between hardware and software have become less clear over the years. Technologies such as FPGAs, virtual machines, APIs, and microcoded instruction sets complicate the simple picture of programs directly manipulating hardware. Second, the recent Turing Centenary has reminded us that the founding insight of theoretical computer science is that hardware and software are, from the viewpoint of computability, almost entirely in-

terchangeable. Still, a primary consequence of this insight has been to allow theorists to ignore issues of platforms and architectures entirely, further reinforcing our sense of programs and algorithms as creatures of pure logic unsullied by materiality.

Software in Museums and Archives

Museum curators and exhibit planners are the people challenged most directly by the special nature of software. Exhibiting computer hardware is not so different from exhibiting a stone axe head or a stuffed Dodo. You put the object in a glass display case, next to an identifying label. Visitors peer briefly at it as they walk by, and a few stop to glance at the text. The objects are arranged to tell some kind of story as the visitor walks. Usually it is a story of progress over time, and so the visitor notices objects in the cases becoming prettier or more complicated the closer he or she gets to the gift shop. These days there tends to be more focus on the story and less on the clutter, while interactive screens are supplementing wordy explanations. The stuffed Dodo and the mainframe would both benefit, should space and funding permit, from being placed in a diorama representing their natural habitat.

You cannot put a soul in a display case. Curators at leading museums such as the Science Museum in London and the Deutsches Museum in Munich have long been aware of the importance of software and have been grappling for a while with the question of how to collect and display it. Traditional approaches are not very satisfactory. One could line up cases full of disks, tapes, and shrink-wrapped boxes to represent the mass-market products of the late-1970s and 1980s, but this would not tell us much about the software itself and would not work at all for early software, enterprise software, internally produced software, or today's downloaded applications.

The challenge is daunting, which is why despite years of discussion no major museum has attempted a full-scale exhibit on the history of software. Even the Computer History Museum in Silicon Valley included relatively little on software in its recently unveiled permanent exhibit. To remedy this the museum is now working on a supplemental exhibit focused on software, and it will be interesting to see what solutions its curators come up with.

Archiving software presents its own challenges. Even if the medium holding the code is preserved then it is far from certain the bits will still be readable decades from now (a particular problem for magnetic media) or that any system able to run the software will still exist. A growing community of enthusiasts has developed online repositories, emulators, and physical archives to address these issues systematically. Coverage is pretty good in areas where collection is easy and nostalgia rampant, such as video games where Henry Lowood at Stanford University has built a substantial archive and most classic titles can easily be downloaded and played from amateur collections of dubious legality. Things like accounting software or online systems are much less likely to survive.

Conceptual Challenges

Thinking about the challenges involved in displaying and archiving software makes me glad that, as a historian who works on books and articles rather than exhibits, I can dismiss them as fascinating problems for someone else to solve. We historians of technology like to think of ourselves as storytellers, writing narratives in which technology, in one way or another, serves as our protagonist or plays a major role. This sidesteps some of the practical problems faced by curators and archivists but, alas, raises a whole set of new problems. The most important of which is: What is software anyway?

Existing historical writing on software has focused on just a handful of topics. One has been the early history of software engineering, particularly the seminal NATO Software Engineering conference held in the Bavarian Alps in 1968. Another has been the history of the software industry, given a thorough overview by Martin Campbell-Kelly in his book *From Airline Reservations to Sonic the Hedgehog*. There has also been a significant body of work about programmers, the people who produce software. Finally we have a significant number of historical accounts of particular software technologies, such as programming languages, mostly written by the people responsible for creating them.

What we do not have, as yet, is a history of software itself; a history of software as a thing or—as a museum curator might call it—an artifact. Such a history might appeal far beyond the (rather small) group of people who think of themselves primarily as historians of information technology. Enthusiasm among humanities and media studies scholars to come to grips with software has recently produced a somewhat dizzying range of self-proclaimed fields and movements, including software studies, critical code studies, video game studies, and demoscene studies. While these identities are still in flux, and media theorists can be rather fickle in chasing after hot new topics, they indicate a broad interest in understanding software as a new and complicated kind of artifact.

Perhaps the most relevant new agenda comes from "platform studies," launched by Ian Bogost and Nick Montfort with their beautifully crafted book *Racing the Beam*, an analysis of Atari's VCS game console that was a fixture in American basements and living rooms during the late 1970s and early 1980s. The focus on platforms recognizes the fact that software without hardware is nothing; as is hardware without software. Each gives meaning to the other. This is particularly apparent to programmers of the VCS, whose 128 bytes of memory could not accommodate a bitmapped screen. Game code was timed precisely against the speed at which the television's electron beam moved across its screen. Thus a program would structure all its activity around the need to place the bits representing the next screen line into the register controlling video output before the TV began to draw that line. This is an extreme case, but it reminds us that platforms are the most enduring technological systems in the world of computing, and that all software is shaped by the constraints of one platform or another. The platform approach also holds out the prospect for meaningful engagement between historians and other kinds of humanities and media scholars.

Existing historical writing on software has focused on just a handful of topics.

Software as Package

Back to software itself. The challenge facing historians is to find ways to tell stories about software that illuminate the fascinating and mysterious nature of software artifacts without falling into the opposite trap of narrowing our focus to look exclusively at the code itself. Our preference is generally to reconstruct the ways of seeing the world that made sense to people in the times and places we are writing about, rather than to impose alien perspectives such as those based on present-day concerns or on something more esoteric like postmodern literary theory.

Fortunately the history of software holds just such a concept, hidden in plain sight. That is the idea of software as a package. We still speak of software packages, yet we rarely stop to consider the implications of the idea. It has a long history: people started talking about programs as packages a couple of years before they started calling them software. So while the idea of packaged software has recently been associated with the fading practice of literally putting programs into shrink-wrapped boxes it was around for decades before computer programs became retail goods. In fact the very idea of programs as software is bound up with the idea of packaging, and goes back to earliest occasions on which people started to think about how programs developed with one computer center could be used by another one.

The practice of sharing programs is as old as the practice of writing programs (and older than the practice of executing them—some have claimed as the first computer programs material published in the 1840s to promote Babbage's never-finished Analytical Engine). The very first computers, built during the 1940s, were experimental one-off machines with their own unique instruction sets. That did not stop the creators of EDSAC, the first useful computer designed from scratch around the modern "stored program" paradigm, from building up a library of reusable machine code subroutines, or in 1951 from filling much of the world's first textbook on computer programming with code taken from this library.[a] However, it did mean the code would have to be reimplemented to work elsewhere.

That soon changed. By 1953 scientists and engineers at more than a dozen different sites were programming identical IBM machines. They began to exchange code and collaborate to develop packages jointly, a relationship formalized in 1955 with the creation of the SHARE user group and the development of new social and technical practices around its library of user-contributed programs. It appears to have been within SHARE that computer users began, by 1958, to refer to "packages." Its projects to jointly develop new program suites addressing high-priority areas were often given code names incorporating the words PAC, for example 9PAC for file maintenance and report generation. (A similar convention survived for decades in the world of mathematical software, as evidenced by the famous LINPACK benchmark for supercomputer ranking and many other PACKs for different specialized areas). Programs within the SHARE library followed specific social and technical conventions so they could be combined as needed by the user group's members. These included wiring control boards in a particular way, standardizing operational procedures, adopting common programming tools, and establishing shared coding conventions. So in this case the work of packaging meant transporting not just the code itself but also the tacit human knowledge, machine configurations, programming conventions, and operating practices it relied upon.

The word software gained favor around 1960, initially as a playful

a Wilkes, M., Wheeler, D., and Gill, S. *The Preparation of Programs for an Electronic Digital Computer.* Addison-Wesley, Reading, MA, 1951.

complement to hardware, already well known as a colloquial term for computer equipment.[b] It was sometimes used to describe everything else the computer manufacturer bundled with the computer hardware—perhaps including services, documentation, and other intangibles. In that sense it has its roots in packaging practice. Programs became software when they were packaged, and not everything in the package was code. For much of the 1960s the most commonly accepted definition of software therefore included only systems programs rather than applications, which were usually produced or heavily customized with the organizations where they were used. For example, a glossy 1962 pullout inserted into *Datamation*, a leading computer industry magazine, promoted Honeywell's expertise in software. This was defined as "the automatic programming aids that simplify the task of telling the computer 'hardware' to do its job." According to Honeywell the "three basic categories of software," were assembly systems, compilers, and operating systems.

When computer manufacturers eventually began to "unbundle" their software offerings, that is, charge separately for them, this was part of a broader trend toward packaging code as a good in its own right—literally as a "ware" for sale. Over the 1970s the mainframe packaged software industry developed from a curiosity to a significant market. This growth relied on a legal framework in which the rights of producers are protected, on the acceptance of banks and investors of software as a viable business, on the willingness of accountants to value packages as assets on a software company's balance sheet, on the willingness of customers to purchase something that may contain flaws they are unable to fix, and on the creation of a set of shared cultural understandings such as the difference between a bug fix (free) and an upgrade (usually paid for). None of these things were initially obvious, and each involved a process of collective learning and negotiation between producers and suppliers during which a variety of practices were experimented with to figure out a viable new way of packaging software.

The history of software is much more than just the history of code.

This is a column rather than a book, and there is insufficient space here to explore the many other chapters in the life of the software package such as the first high-quality manufacturer-supported packages (IBM's FORTRAN seems to have been a model), commercial software libraries, the shrink-wrapped model developed by the personal computer industry, online app stores, and subscription services. The shape and size of the package varied, and the bundle of code, documentation, services, support, and tacit knowledge assembled to make an enterprise product like SAP ERP into a salable commodity are clearly quite different from those packaged as Angry Birds. Still, airmail envelopes and modern intermodal shipping containers are both packaging technologies functioning on very different scales.

The point remains that the history of software is much more than just the history of code. Despite its apparent immateriality, software has always been tied to a platform and has always been physically embodied in one form or another. What turned programs into software was the work of packaging needed to transport them effectively from one group of users to another. To understand software we cannot just look at the bits. We need to examine the whole package. C

b Shapiro, F.R. Origin of the term software: Evidence from the JSTOR electronic archive. *IEEE Annals of the History of Computing 22*, 2 (2000) located an initial usage in 1958 by mathematician John W. Tukey to describe automatic programming aids.

Thomas Haigh (thaigh@computer.org) is an associate professor of information studies at the University of Wisconsin, Milwaukee, and chair of the SIGCIS group for historians of computing.

Further Reading

Akera, A.
Voluntarism and the fruits of collaboration. *Technology and Culture 42* (2001), 710–736. Examines the early history of the SHARE user group mentioned in this column.

Blanchette, J.-F.
A material history of bits. *Journal of the American Society for Information Science and Technology 62*, 6 (2011), 1042–1057. Begins with an interesting critique of the idea of information as an immaterial digital substance, as sometimes favored by media theorists, then surveys key modular design techniques used to produce the computer systems that support digital representations of information.

Campbell-Kelly, M.
The history of the history of software. *IEEE Annals of the History of Computing 29*, 4 (2007), 40–51. Explores the development of software history over its first few decades, identifying trends and key works.

Haigh, T.
Software in the 1960s as concept, service, and product. *IEEE Annals of the History of Computing 24*, 5 (2002), 5–13. Examines the origins of the software concept and reconstructs the challenges and opportunities early software packages provided to data processing users.

Hashagen, U., Keil-Slawik, R., and Norberg, A.L., Eds.
Mapping the History of Computing: Software Issues. Springer-Verlag, New York (2002). Papers and discussion from leading scholars chosen to reflect different approaches to software history. Includes several insightful papers from museum specialists on the challenges of collecting and displaying software.

Lowood, H.E.
The hard work of software history. *RBM: A Journal of Rare Books, Manuscripts, and Cultural Heritage 2* (2001), 41–161. Overview of the challenges of software history, including those posed to curators.

Mahoney, M.S.
What makes the history of software hard. *IEEE Annals of the History of Computing 30*, 3 (2008), 8–18. A thoughtful attempt to position software history within an approach to what Mahoney called the "histories of computing(s)" structured around user communities and practices rather than hardware.

Williams, R. and Pollock, N.
Software and Organizations: The Biography of the Enterprise-Wide System or How SAP Conquered the World. Routledge, London, 2009. Applies perspectives from science studies to enterprise software packages, thus making a case for examining their "biographies" to understand them.