Software in the 1960s as Concept, Service, and Product

Thomas Haigh
Colby College

Packaged application software established a small but important corporate niche during the 1960s. The author charts the shifting meaning of the word *software*, situates the first software companies within the overall computer services market, and probes the attractions and limitations of the first packages from the viewpoint of their potential purchasers: managers of data processing.

The term software is newer than most of the things that it is today used to describe. Many data-processing pioneers assumed that applying their electronic computers to specific administrative tasks would be straightforward. While they realized that the new machines would require programming, many expected this to be a one-time, rapid, and easily compartmentalized job, firmly subordinate to the larger task of business analysis. They soon discovered that application program creation was costly, difficult, and ongoing. By the mid-1950s, they had come to care a great deal about programming.[1] But only around 1960, however, would a well-informed data-processing manager have nodded knowledgably if *software* came up in conversation. During the 1950s, the term was not used, although *hardware* was already well known as a colloquial term for computer equipment. When *software* did achieve currency, it was as hardware's complement, describing everything else the computer manufacturer provided. This ensured that term's widespread, if ill-defined, use.[2]

Software: A fluid concept

By the mid-1970s, accepted linguistic usage of *software* had shifted toward "programs and other operating information used by a computer" offered by today's *Concise Oxford English Dictionary*. Software became a synonym for computer program, excluding perhaps only a processor's microcode and the firmware burned into permanent memory. However, for much of the 1960s, *software* was commonly understood in a narrower sense, as what was later called systems software—programs used to construct other programs, or operating systems to control computer hardware. During the early 1960s, computer manufacturers dramatically stepped up their efforts in this area. IBM, for example, had gradually broadened the range of system tools it shipped—from none with the first 701s, to symbolic assemblers and loaders with the 650 and 704, to increasingly complex input–output (I/O) and control programs with its second-generation transistorized machines. Cobol's arrival challenged every computer manufacturer to produce compilers for this complex high-level language, an effort that frequently strained the state of the art to the breaking point and beyond.

A 1962 Honeywell advertisement touted the firm's expertise in this new field, defining software as "automatic programming aids that simplify the task of telling the computer 'hardware' to do its job" and observing that the "three basic categories of software" were assembly systems, compilers, and operating systems.[3] This was clearly what a *Datamation* editorial writer had in mind, when observing that

> the well-publicized potentials of software have been riddled during the past few months with a barrage of generally well-aimed criticism. The effect has been one of withdrawn embarrassment; a quiet revision of delivery schedules; a crash program to check out a compiler or two, and some judiciously phrased, user-directed pleas for compassion and patience.[4]

Likewise, when Asher Opler, then responsible for programming at Computer Usage Corporation, published a 1964 article in *Datamation* on the "Measurement of Software Characteristics" he confined his attention entirely to "automatic programming and operating systems (software)." Opler suggested that these programs so influenced a computer's real performance that it was absurd to continue to evaluate a computer's suitability for an application on the basis of its hardware capabilities alone.[5]

Software's other implicit definition at that time, broader but equally unfamiliar today, included not only systems tools but also any combination of tools, applications, and services purchased from an outside vendor. This ambiguity was captured at the 1963 meeting of Fred Gruenberger's informal "Rand Symposium." The symposium took place every year immediately before the Joint Computer Conference, giving an invited crowd of computing and data processing luminaries a chance to discuss the issues of the day. When the possibility of making software production an engineering activity was raised, participants discovered that they lacked a common definition of software. While one participant suggested that "many of us refer to software as programs to be used by programmers," Frank Wagner, recently moved from North American Aviation to fledgling software company Informatics, countered that

> I always use the term to mean anything that isn't clearly hardware or engineering design.... not only applications programming, but the writing of specification for programs, the giving of advice to people who might want to use computers, the installation manuals, etc.[6]

Wagner was a prominent member of the southern California computing community, having played important roles in both the SHARE user group and the Association for Computing Machinery. Bob Patrick, a well-known California computer consultant, then recently departed from leading computer services firm CEIR (formerly Council for Economic and Industrial Research), took issue with this definition. What he challenged was not the inclusion of nonprogramming services but of application programs, saying "I don't believe that application packages are software," regardless of who developed them.[6]

A very broad definition of software was endorsed as late as 1967 by *EDP Analyzer*, Richard Canning's authoritative newsletter. When it repeated a claim by Walter F. Bauer of Informatics that the software market share held by independents (companies that did not make computers) had risen from 1 percent in 1960 to 3 percent in 1965 and would reach 10 percent by 1970, it also quoted Bauer's software definition: "systems analysis and design, programming and computer-based services, accomplished by users, computer manufacturers and others." Bauer argued that independent producers were rapidly shifting focus from systems software toward applications programs, which he estimated would rise to about 50 percent of their revenues.[7]

As real-time systems expert Robert V. Head aptly observed the next year, "The term 'software'... is subject to redefinition from time to time and to varying degrees of individual interpretation." His own definition included

> not only the series of program instructions needed to direct the computer to do a particular job, but also ... the entire process of systems analysis and design, programming, testing and implementation, as well as the documentation that accompanies this process.[8]

Not all "software" was programs, and not all programs were "software." According to some of these definitions, at least, a given piece of code might have been considered software if obtained from another firm, but simply a program if written in-house. Yet a contract for an external group to perform programming work of any variety,—even testing or documentation—might have been considered a software project. Because of these conceptual shifts, contemporary readers may easily misunderstand the intended meaning of many sources from the 1960s. Early discussions of problems in the "production of software," or the "software industry," were invariably referring to something other than what we would today understand by these terms. For example, much early discussion of a "crisis" in software or of software development problems was actually concerned entirely with systems software.[9] Likewise, complaints about the rising proportion of computer costs attributed to software, or the problems of software project management, generally referred to the hidden costs of systems software bundled with computer hardware, not to the ever-present cost of in-house application programming. In contrast, estimates of the software industry's size and future growth usually referred to the entire market for separately purchased computer-related services and code.[10]

Toward the packaged application

Those, like Bauer, who favored a broad definition of software essentially used the term to describe any externally provided code or services. This conflation of independently produced programs and independently offered programming, analysis, and advice made considerable sense during the early 1960s. There was, as yet, no such thing as a shrink-wrapped application package. Most application programs were written in-house from scratch, although firms increasingly relied on operating systems and programming tools from manufacturers. Early interest in packages stemmed from the reasonable, generally held idea that it might be more effective to take an existing program and modify it than to write an entirely new one. Computer manufacturers supplied such "canned" applications free with their machines. A few of these packages enjoyed considerable success, such as IBM's early integrated package '62 CFO, which business historian JoAnne Yates has shown eventually found hundreds of users among medium-sized insurance companies. According to Yates, many of these companies would have been unlikely to order the relatively inexpensive IBM 1401 computer it ran on had the package not lifted most of the programming burden from their data-processing staff.[11]

Many of the earliest software firms relied on governmental, particularly military, contracts. The importance of RAND Corporation spin-off SDC to computing's early history and to the SAGE project is well known. Martin Campbell-Kelly has suggested that large firms producing highly complex systems of this kind satisfied a market almost entirely distinct from the needs of mainstream business for low-cost application packages. Many offered computer services as part of a broader range of scientific or technical services. The Planning Research Corporation (PRC), for example, was founded in 1953 by three scientists, but a stream of defense contracts helped it grow to 507 professional employees by 1965, more than half of whom worked on computer-related issues. (Despite this, the firm had yet to install a single computer; it rented computer time as needed.)[12]

The largest software companies (in this broad sense) included PRC, the Computer Sciences Corporation (CSC), CEIR, Computer Applications Inc., and Informatics. These firms were of very different character from Microsoft and other firms that spring to mind when software is mentioned today. Their work was closer to the consulting and services tasks today undertaken by accounting firms, independent specialists like AMS and EDS and computer vendors like IBM. Then, as now, small and flexible companies satisfied most of the demand for computer services. Barriers to entry were low, and stock options lured skilled programmers to the start-up firms. A spate of initial public offerings around 1966 led investors to pour money into the field, which was soon crowded by an estimated 2,000 companies.

Independent software companies did not generally attempt to compete head-to-head by selling their own skeleton application code for companies to adapt. They did, however, undertake similar projects for a number of different customers, building a library of reused routines, acquiring expertise, and finding themselves able to complete additional projects better and cheaper. After a certain point, custom applications with recycled code gradually evolved, into something more like standard-packages. It was, however, the appeal of getting a truly tailored solution rather than a generic program that had steered customers away from the computer vendor toward the software company in the first place.

By the mid-1960s, application packages were becoming better established. *EDP Analyzer* reported that

> The change in interest in application packages became evident [in 1965]. While some interest had existed previously, it appeared to be quite local. The arrival of the general purpose inventory forecasting packages may have been the triggering influence for the growing popularity of packages.[13]

Inventory management was a most popular application for data-processing departments looking to do more than simply automate clerical jobs such as payroll. The operations research aspects of this task, however, posed considerable challenges to corporate programming staffs.[13]

The first independently produced programs to be licensed as standard packages apparently were systems software such as file management and report generation utilities. These filled niches left empty by the computer manufacturers' own software. Perhaps the most important early independent software package was Informatics' Mark IV file management system, which like many early packages, evolved from work begun as a custom development project. First offered in 1967, it helped propel Informatics to become a leading 1970s' software firm. Informatics, as we have seen, had a much broader view of software and continued to derive most of its revenue from custom programming and consulting work. As the 1960s closed, Bauer suggested that, despite earlier optimism, the total market for packaged software constituted only about 10 percent of that for contract development work.[14]

In 1968, when *Business Automation* magazine interviewed him, Head was already established as a leading software expert. He had worked on the pioneering ERMA (electronic recording method of accounting) bank automation and SABRE airline reservation systems in a career that had already involved working for GE and IBM, in the banking industry, and in senior IT-related positions for consulting firms Touche, Ross, Bailey & Smart and CSC. Head had just formed his own company, the Software Resources Corporation, which he discusses elsewhere in this issue. He suggested that "a sort of software explosion" had occurred in 1967, caused by a shift toward more complex management information system (MIS) applications, a change that demanded a level of programming and system design skills rarely present in corporate data-processing departments. Head, who, given the breadth of his own work experience, was well placed to know, suggested that "at this point virtually no one is making a substantial profit on packaged software, although the potential seems very great."[15]

The assumption that hardware and operating systems should be standard for all users while application software should be altered for different industries was not yet hard and fast. Head believed that hardware, programming languages, and operating systems would prove more efficient when adapted to industry sectors such as banking or retailing. Odd as this seems given present experience, we must recall that the construction of early real-time applications such as SABRE had required the building of specially designed operating systems—in this case, by a collaboration between IBM and American Airlines.

Such operating systems were closely optimized for the programs they would run alongside. The first operating system for the IBM 701/704 was produced by the SHARE user consortium. Some firms tackling real-time business applications coded their own operating systems. Even when firms used vendor-supplied operating systems, the operating system had often been tweaked or patched by individual users. Some companies even tried to write their own operating systems for System/360 computers, often in conjunction with a plan to build an integrated, online MIS.[16]

The line between programming language and application software was also unclear. In a 1965 address to a combined meeting of San Diego data-processing societies, Paul H. Rosenthal of CSC suggested that application packages were a blind alley:

> There is actually no extensive production being done today utilizing application packages .... These types of packages have not been widely accepted, and outside of some very selected industries or functional areas are not considered by most people to be the answer to reducing the cost of applications programming.[17]

Citing the success of generalized file management systems and specialized language compilers in the scientific field, he suggested that the long-term answer was not "packages as they are currently constructed" but new methodologies coupled with "application compilers" able to go straight from requirements to code. Within two years, he expected these systems to let line managers write their own applications as needed via teletype. This would facilitate construction of an MIS and eliminate traditional application programming, while giving data-processing specialists "a far higher status in the total managerial hierarchy." Skepticism, he insisted, was "no longer possible."[17]

There was thus no absolute line between systems software and applications programs, and neither was it clear that the former would invariably come from the hardware manufacturer and the latter from in-house or independent efforts. Nor was it clear that most companies seeking access to an externally produced program would run it on their own hardware or operate it using their own personnel. From the earliest days of computing, firms such as payroll-processing giant ADP had sold bundles of services based on proprietary software. From the mid-1960s, facilities management operations such as Ross Perot's EDS were eager to operate computer installations under contract. In some cases, this included programming services and standardized packages. As interest in networking and remote access to computers increased, many expected these models to become the norm. Time-sharing, a means of giving several users real-time access to a single computer, was viewed as the basis on which "computer utilities" would be built. Under this model, thousands of users would subscribe to giant networks, using terminals to access hardware and software running on remote computers. Well over a hundred companies rushed to enter the time-sharing business.[18]

Time-sharing services were initially popular with scientists and engineers, who liked the convenience of interactive computers. The services offered libraries of useful programs and subroutines to assist with calculations. As the market became crowded toward the end of the 1960s, and time-sharing companies targeted the potentially larger administrative computing market, they sought to differentiate themselves by offering applications packages rather than just computer time. A firm would effectively lease applications as a package, with computer hardware, software, and services thrown in. GE (the market leader) offered a package to banks; BBN (a computer services firm now best known for its contract work on Arpanet), a package for architects; and Univac had ambitious plans to build nationwide networks to serve specific industries. Its integrated package of billing, accounting, inventory, and other operations was offered initially to the wholesale wine and liquor industry.

While this was ultimately not a sizable market, continued attention to the concept left managers uncertain whether the application packages of the 1970s would be installed on their own computers or rented through time-sharing.[19]

User views on software packages

How did the corporate data-processing managers of the 1960s feel about packaged application software? A survey of the magazines and journals most closely associated with this community (*Business Automation*, *Datamation*, and *EDP Analyzer*) offers hints, and several important articles from these sources are discussed below. From 1967 onward, a small but growing number of subscribers could turn to the International Computer Program directories (discussed elsewhere in this issue). But the most important evidence is negative—akin to Sherlock Holmes' realization that a dog had failed to bark during the night. While a good number of articles discussed the software industry's boom, and time-sharing's revolutionary potential, there were few nuts-and-bolts treatments of how to purchase or evaluate application software packages. Throughout the decade, packages remained relatively unimportant as a source of application programs, or even as part of the overall software and services market.

Like anyone keeping one eye on the business press and one on the stock market, data-processing managers were exposed to an enthusiasm for the software industry, which had quite outstripped its accomplishments. As Head observed in 1970, at the tail end of the first market bubble in software stocks,

> For a long time now, the software industry has been a prime concept of interest on the part of investors, with resultant enormously inflated stock values. Price-earnings ratios of eighty to one seem normative, and infinity to one not unusual, in the case of numerous companies that have never turned a profit. There has been almost a compulsion to 'go public' on the part of many marginal firms, and millions of dollars have been obtained from the public through such offerings.[20]

One remarkably enthusiastic statement came in 1970, from Pabst Brewing's data-processing director. The author claimed that the new generation of packaged applications was much superior to the old, manufacturer-supplied packages. As a result, "Low supply and high demand for in-house systems and programming talent may not be as long lived as many think." He seized on packaged application software as one in a long line of technologies destined to end corporate application programming as then known. (Earlier technologies of this kind included "automatic coding," high-level languages, and decision tables). The data-processing department's entire analysis, programming, and maintenance function could, he claimed, be replaced with three people: a data-processing coordinator, a software/applications analyst to choose packages, and a single package-modification programmer to configure them. This enthusiasm apparently reflected a utopian dream rather than hard-won experience.[21]

The programming of administrative applications remained an activity performed largely by a company's own data-processing staff. In 1969, the *Wall Street Journal* surveyed almost 800 executives with responsibility for computer procurement and found that 44 percent claimed all programs used were written in-house; another 38 percent said that many were. Even among the largest companies (the sector identified by the survey as the primary users of what it called "Programming [Software] Services"), just 45 percent of respondents to this question reported any purchase of outside programming (a category including both packages and services). These same companies relied more heavily on bundled programs supplied by the manufacturers of their computers, utilized by 80 percent of responding firms.[22]

Any software acquisition—packaged or custom; system or application—ultimately became a set of make-or-buy questions. No package entirely removed the need to "make" some elements of the system, but it was sometimes possible to reduce this burden. Unlike the hopeful schedules and fuzzy definitions associated with in-house development, packaged software was a known quantity. Its costs and capabilities could be measured. In addition, many companies found their programming and analysis teams chronically overworked, with a large backlog of urgent tasks. Packaged software promised to alleviate this.

During the second half of the 1960s, most data-processing departments were engaged in a transition from second-generation machines (such as the IBM 1401 or 7040 series) to third-generation machines such as those in the System/360 range. In the process, they used complex operating systems, large disk drives, and terminals for the first time. Since it would take a major programming effort to reimplement their existing systems for this new environment, the transition to third-generation systems was an obvious moment at which to consider the shift to packaged applications.

This approach held the additional promise that different applications from the same vendor might be easier to integrate than in-house packages that had never been properly coordinated. CSC, for example, offered a number of generalized applications to banks to handle common applications such as payroll, personnel records, stockholder records, and general ledger. Such application~~s~~ packages were somewhat generalized—modification and maintenance would still be necessary—but more flexible than a typical program. While CSC charged each customer when new options and capabilities were delivered, bug fixes were free of charge and only a modest fee was levied for the basic updates to meet new legal regulations. As a result, it was realistic to hope that use of a package would dramatically cut the cost of program maintenance.[23]

In 1968, Head used a *Datamation* article to give data-processing managers what was then their most detailed, practical guide to packaged software acquisition. He identified application-package sources as manufacturers (in whose products he had little faith), user groups (some of which maintained extensive lists of user-submitted software), and software companies. Whatever the source, a package would likely require substantial modification. A well-designed commercial package could easily pay for itself by lowering the overall project cost, especially if this also led to reduced maintenance costs. Head suggested that commercial applications generally cost between $2,000 and $20,000, representing 10 or 20 percent of the original development cost. While acknowledging that "[t]here may still be some resistance on the part of data processing managers accustomed to obtaining software 'free' from the computer manufacturers," he argued that such reliance might prove a false economy.[24]

Head's advice for the evaluation of software packages was more pragmatic. He suggested that data-processing managers assign their own weights to each of the following factors and score packages under consideration accordingly:

- package cost (including indirect costs for modifications, training, installation, conversion, running, maintenance, and so on),
- package quality,
- design features (file organization, control and audit features, programming techniques, flexibility, and so on),
- generality ("among the most important package criteria"),
- expandability (although an overgeneralized package might prove inefficient),
- operational status (how extensively used and bug-free a package is),
- equipment configuration needed to run it,
- programming language it is written in,
- documentation (separate descriptions required on each system level: program, operations, and user),
- installation support (must be agreed in advance; may include file conversion and training assistance), and
- maintenance ("the acquisition terms for a package should include some assurance of error-free operation for a reasonable period of time").[25]

The list demonstrates that software acquisition remained complex. The importance of the programming language used, program documentation, and program expandability illustrate that an application program was subject to extensive customization. Programming techniques of the era made it hard to produce a program that would run on less-powerful hardware while taking advantage of larger memories, disk drives, and the like when available. Similarly, unnecessary features might slow down a program and make it harder to maintain. As Head warned in a later guide to the same topic, computer staff might not prove a package's most reliable judges. While a manager was to some extent at the mercy of their judgments, he or she must also remember that their "professional pride" might lead them away from cost-benefit calculations toward the conviction that they could produce a better package. Said Head: "Experience has shown that an outside package, no matter how estimable, can be torn asunder by an astute technician bent on ferreting out and magnifying all possible deficiencies."[26]

In a short 1971 book, Head surveyed the state of the software industry at that time. He found that some packaged application programs were supplied in Cobol, for compilation on a range of machines from different manufacturers. Others were more closely tied to particular machines and operating systems. His analysis of the market for payroll programs, a leading application of the period, was particularly informative. Most such programs cost about $20,000. Two of the most successful appeared to have sold about 75 copies each. One of these, the CSC payroll system, was slow and required a large, 65-Kbyte memory. It was relatively easy to use and install, reducing the need for skilled computer personnel. However, this usability came at the expense of flexibility—reports not needed for a particular payroll run, for example, could not be turned off.[27] Such programs continued to appeal primarily to companies without the staff, money, or time to develop a better system of their own.

IBM's decision, in 1969, to unbundle its software from its hardware is usually considered crucial to the establishment of a viable market for packaged application software.[28] While undoubtedly important, this event appears to have reinforced shifts already under way, and its effects were not felt immediately. In the early 1970s, most application software was still produced by user organizations; most of the packages used continued to come from IBM and other computer manufacturers. While the entrenchment of the System/360 architecture and the acceptance of standardized high-level languages (most particularly Cobol) provided a larger potential market than ever before, this market remained largely untapped. In his 1971 book, Head suggested that although "the potentially great profits" were associated with a

> potential market for these systems in the tens of thousands ... even highly successful packages have at this point sold only in the neighborhood of fifty systems with a mere handful of outstanding success stories claiming sales in the hundreds.[29]

In the subsequent decade, firms such as Management Science of America (MSA) and the University Computing Corporation (UCC) finally managed to nurture a market for packaged applications software that grew steadily and proved profitable. (The largest and most successful firms, however, still tended to specialize in systems software.) Change in computer usage lagged expectations, as firms ported their applications from one machine to another, using emulation and continual patching to support ancient application code on new hardware.

The challenge in application program acquisition has always been to optimize the fit of a system to the requirements of a particular business while minimizing the amount of specially written code and lowering the cost of ongoing maintenance. Packaged applications ultimately played an important part in reconciling these goals, as part of a broader repertoire of complementary techniques including generalized I/O routines, high-level languages, structured programming and design, CASE (computer-aided software engineering) and RAD (rapid application development) tools, database management systems, and code reuse. (The latter eventually gave rise to object orientation.) These techniques were supported by substantial improvements in operating system technology and computer architecture. Meanwhile, software packages and services remained closely intertwined in how they were bundled, sold, and used.[30]

We cannot hope to understand software's early history without understanding the work done inside user organizations to adapt and supplement packages. A narrow understanding of software, based on experience of the shrink-wrapped PC-software market of the 1980s, will not help here. Indeed, such a conception might instead mislead the contemporary analyst as well as the historian. Into the 1980s, Informatics continued to derive more of its revenues from custom development and consulting services than from packaged software sales. Even today, the shrink-wrapped model applies only to a small part of the business software world, and many analysts believe it to be in terminal decline.[31] Installing complex packages such as those offered by SAP, for example, requires a tremendous configuration and customization effort. Moreover, the proliferation of programming tools such as databases, object technology, and rapid application development systems has been responsible for the production of more custom application software, not less.

A better understanding of the origins of packaged software enriches our understanding of its likely future in at least two ways. First, it reminds us that different models coexist and that packaged software, custom programming, and consulting services are complementary. Services are usually inseparable, most programming effort is still devoted to custom applications, and the transition from custom development to package remains a continuum. Second, a better understanding teaches us that change is invariably slow and incomplete and that no new model—be it time-sharing in the 1960s or Internet-based application service providers today—can be expected to fill all niches or to sweep away existing technologies overnight. Had the technology investors and entrepreneurs of the past few years paid more attention to history, their dollars and their dreams might not have been lost to it with quite such rapidity.[32]

References and notes

1. On the early history of data processing and the role of programming within it, see T. Haigh, "The Chromium-Plated Tabulator: Institutionalizing an Electronic Revolution, 1954–1958," *IEEE Annals of the History of Computing* (hereafter called *Annals*), vol. 23, no. 4, Oct.–Dec. 2001, pp. 2-31.
2. For discussion of the earliest known use of "software" in the context of computers, see F.R. Shapiro, "Origin of the Term Software: Evidence from the JSTOR Electronic Archive," *Annals*, vol. 22, no. 2, Apr.–June 2000, pp. 69-71. Shapiro discusses a 1958 article by mathematician John W. Tukey, who used the term to describe automatic programming aids (such as compilers and assemblers) of the type provided by computer manufacturers.

3. Honeywell, "A Few Quick Facts on Software," *Business Automation*, vol. 7, no. 1, Jan. 1962, pp. 16-17.
4. "Software on the Couch," *Datamation*, vol. 7, no. 11, Nov. 1961, pp. 23-24. A similar definition of software as a "programming system package" can be seen in H.R.J. Grosch, "Software in Sickness and Health," *Datamation*, vol. 7, no. 7, July 1961, pp. 32-33. See also "The Master Plan for Kludge Software," *Datamation*, vol. 8, no. 7, July 1962, pp. 41-42.
5. A. Opler, "Measurement of Software Characteristics: Evaluation Techniques," *Datamation*, vol. 10, no. 7, July 1964, pp. 27-30.
6. F. Gruenberger, "Rand Symposium 6," in Rand Symposia Collection (CBI 78), Charles Babbage Institute, Univ. of Minnesota, Minneapolis (hereafter, CBI), 1963. This discussion in many ways precipitated the much better known NATO Conference on Software Engineering held in 1968 and 1969. Participants discussed parallels between hardware and software, the difficulty of managing programmers, the importance of literary style in programming, and the applicability of engineering techniques to software. Participants included Barry Gordon, Bob Patrick, Richard Hamming, Francis V. Wagner, and M.D. McIllroy.
7. Bauer offered his definition during a Fall 1965 address to the Los Angeles chapter of the ACM. See "Independent Software Companies," *EDP Analyzer*, vol. 5, no. 11, Nov. 1967.
8. For Head's discussion and his own definition, see R.V. Head and E.F. Linick, "Software Package Acquisition," *Datamation*, vol. 14, no. 10, Oct. 1968, pp. 22-27.
9. See particularly H.R.J. Grosch, "Software in Sickness and Health," *Datamation,* vol. 7, no. 7, July 1961, pp. 32-33, and "Software on the Couch," *Datamation*, vol. 7, no. 11, Nov. 1961, pp. 23-24. One of the earliest articles to imply our modern definition of "software costs" as including in-house development of application programs is H.A. Lustig, "The High Cost of Software," *Business Automation*, vol. 13, no. 5, May 1966, pp. 37-38. Conversely, most of the crises attributed to data processing in this period had little direct relation to software—see D.R. Daniel, "Management Information Crisis," *Harvard Business Rev.*, vol. 39, no. 5, Sept./Oct. 1961, pp. 111-121, or A.E. Keller, "Crisis In Machine Accounting," *Management and Business Automation*, vol. 5, no. 6, June 1961, pp. 30-31, for examples.
10. In perhaps an extreme example, one author presented service bureaus, contract programming, consulting services, personnel services, package suppliers, proprietary software services, facilities management, "dedicated applications companies," time-sharing services, providers of turnkey systems, and educational services as the 11 components of the "software service" market. See F.A. Frank, "Software Services: An Outside Outlook," *Business Automation*, vol. 16, no. 11, Nov. 1969, pp. 55-61.
11. J. Yates, "Application Software for Insurance in the 1960s and Early 1970s," *Business and Economic History*, vol. 24, no. 1, Fall 1995, pp. 124-126.
12. M. Campbell-Kelly, "Development and Structure of the International Software Industry," *Business and Economic History*, vol. 24, no. 2, Winter 1995, pp. 73-110. On the Planning Research Corp., see "The Changing Software Market," *EDP Analyzer*, vol. 4, no. 7, July 1966.
13. "Application Packages: Coming into Their Own," *EDP Analyzer,* vol. 5, no. 7, July 1967.
14. For the Mark IV, see M. Campbell-Kelly, "Development and Structure of the International Software Industry," *Business and Economic History*, vol. 24, no. 2, Winter 1995, pp. 73-110; J.A. Postley, "Mark IV: Evolution of the Software Product, a Memoir," *Annals*, vol. 20, no. 1, Jan.-Mar. 1998, pp. 43-50; and R.L. Forman, *Fulfilling the Computer's Promise: The History of Informatics, 1962–1982*, Informatics General Corp., 1984.
15. "The Software Explosion," *Business Automation*, vol. 15, no. 9, Sept. 1968, pp. 24-29. Head went on to help found SMIS, the Soc. of Management Information Systems (known today as SIM), and spent the latter part of his career in senior IT positions with the US federal government and as a consultant in the same area.
16. Head earlier detailed his concept of industry-specific computing platforms in R.V. Head, "Old Myths and New Realities," *Datamation*, vol. 13, no. 9, Sept. 1967, pp. 26-29. For an example of a homebrew 360 operating system, see L.F. Zaino, "How Does an Operating System Work?," *Systems & Procedures J.*, vol. 19, no. 1, Jan./Feb. 1968, pp. 20-23. On the history of MIS and its relationship to the promotion of third-generation computers, see T. Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950–1968," *Business History Rev.*, vol. 75, no. 1, Spring 2001, pp. 15-61.
17. P.H. Rosenthal, "Future Programming of Computer Applications," *Systems & Procedures J.*, vol. 18, no. 1, Jan./Feb. 1967.
18. Although the computer utility concept is generally believed to have originated with M. Greenberger, "The Computers of Tomorrow," *The Atlantic Monthly*, May 1964, pp. 63-67, it goes back at least five years earlier to the 1959 conference presentation published as A.O. Mann, "A Publicly Regulated System of Management Control Services," in *Management Control Systems*, D.G. Malcolm and A.J. Rowe, eds., John Wiley & Sons, New York, 1960, pp. 245-263. For

an early discussion of its relevance to data processing, see R.E. Sprague, "The Information Utilities," *Business Automation*, vol. 12, no. 3, Mar. 1965, pp. 42-47.
19. E.H. Menkhaus, "Time Sharing is Everybody's Thing," *Business Automation*, vol. 16, no. 9, Sept. 1969, pp. 26-35 and p. 38; and R.V. Head, *A Guide to Packaged Systems*, Wiley-Interscience, New York, 1971. Others believed that the crucial advantage of the computer utility would come with the automatic interchange of orders, payments, and other information between firms that would follow once different businesses used shared computers to hold their information. These feelings were only reinforced in 1970, as a collapse in the stock market bubble for software and time-sharing firms caused scores of start-up firms to begin unraveling. As *Business Automation* reported at the time, the future of time-sharing was seen to lie not in the general-purpose computer utility but in the provision of "industry-oriented software on a national basis …. Time sharing has become an information transfer business and not a computation or calculation business." See E.J. Menkhaus, "Time Sharing: More Glitter than Gold," *Business Automation*, vol. 17, no. 11, Nov. 1970, pp. 36-42.
20. R.V. Head, "Twelve Crises—Comments on the Future of the Software Industry," *Datamation*, vol. 16, no. 3, Mar. 1970, pp. 124-126.
21. J.E. Hackney and N.L. Paul, "The Wheel Exists," *J. Systems Management*, vol. 21, no. 5, Nov. 1970, pp. 40-41.
22. *Wall Street J.*, "Management and the Computer: A *Wall Street Journal* Study of the Management Men Responsible for their Companies' Purchases of Computer Equipments and Services," Data Processing Management Assoc. Records (CBI 88), CBI, 1969.
23. "Application Packages: Coming into Their Own," *EDP Analyzer*, vol. 5, no. 7, July 1967.
24. R.V. Head and E.F. Linick, "Software Package Acquisition," *Datamation*, vol. 14, no. 10, Oct. 1968, pp. 22-27.
25. Ibid., pp. 25-26.
26. R.V. Head, *A Guide to Packaged Systems*, Wiley-Interscience, New York, 1971.
27. Ibid., pp. 41-43.
28. Unbundling has been the subject of considerable historical inquiry, including several articles in this issue. For contemporary reaction, see A. Dratell, "Unbundling: The User Will Pay for the Works," *Business Automation*, vol. 16, no. 8, Aug. 1969, pp. 36-41.
29. R.V. Head, *A Guide to Packaged Systems*, Wiley Interscience, New York, 1971, p. 66.
30. For a critical discussion of the largest mainframe software companies at the start of the 1980s, see S.T. McClellan, *The Coming Computer Industry Shakeout: Winners, Losers, and Survivors*, John Wiley & Sons, New York, 1984, pp. 240-263.
31. Some software remains in the public domain, particularly in the Linux area where some companies attempt to give away the code while charging for support and services. In the server and enterprise software markets, software is leased annually rather than purchased and is often priced together with service and support contracts. Current industry opinion suggests a general move toward the use of online software (via application service providers—the time-sharing systems of the 21st century) and the annual leasing of desktop application software.
32. On the recent travails of the ASP industry, see C. Koch, "Boy, That Was Fast!," *CIO Magazine*, 15 Nov. 2000, http://www.cio.com/archive/111500/boy.html [current as of 16 Jan. 2002]. For a dismissal of the concept as "rehashed time-sharing service bureaus," see B. Lewis, "Rather than Focusing on Best Technologies, Let's Look at and Learn from the Year's Worst," *Infoworld*, 26 Jan. 2000, http://staging.infoworld.com/articles/op/xml/01/01/29/010129oplewis.xml [current as of 16 Jan. 2002].

Acknowledgments

The author would like to thank Burton Grad and Robert V. Head for their encouragement and contributions and the referees for their comments on the draft version. This research has been supported by fellowships from the Charles Babbage Institute of the University of Minnesota and the IEEE History Center.

Thomas Haigh is teaching at Colby College and completing a PhD in the history and sociology of science at the University of Pennsylvania. His dissertation, *Technology, Information, and Power: Administrative Technicians in the American Corporation, 1917–1975*, is the first full-length synthetic history of the corporate use and management of information technology during that period. He holds a BSc and MEng in systems integration from the University of Manchester, UK. Fellowships include a Fulbright Award, the IEEE Life Member Fellowship in Electrical History, and the Tomash Fellowship from the Charles Babbage Institute.

Readers may contact Thomas Haigh at tdhaigh@colby.edu; http://www.tomandmaria.com/tom.