5.

A BRIEF HISTORY OF THE APPLICATION; OR, HOW THE COMMODIFICATION OF SOFTWARE SHAPED ITS NEGOTIABILITY

5.1 INTRODUCTION

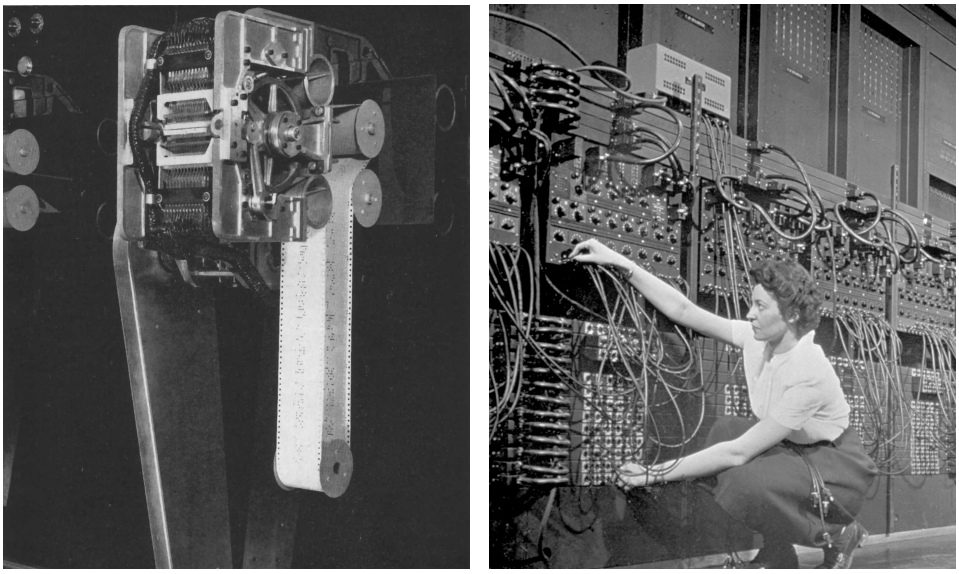
The "application" is not a natural, inevitable design of software, but rather a particular construct profoundly shaped by the process of commodification it went through in the United States between the 1950s and the 1990s. Despite its ubiquity, the application model of software is almost wholly ignored as a topic of inquiry by HCI research and the history of computing communities. As a result, it is difficult to imagine our everyday interaction with computers mediated by anything other than applications, or see a path towards a different model in the future.

The aim of this chapter is to trace how the *negotiability* of software has changed over time by describing the history of the application model from the 1950s until the 1990s. It will follow the social and economic construction of this technology by looking at how, at different points in time, software was defined, what its purpose was imagined to be, who was included in its development, and what activities it supported or inhibited.

The words used to distinguish between different designs and commodifications of software, even the word software itself, are anachronisms that did not exist or refer to the things we might use it for today. For the sake of clarity, however, I will use software ahistorically as a catch-all term for non-material computer instructions, in conjunction with the actual words used to describe the artefact at the time whenever possible.

5.2 THE MATERIAL CREATION OF SOFTWARE

Before the 20th century, there was no need to distinguish between 'hardware' and 'software': a machine's execution instructions were largely hardcoded into the device's design. A first *conceptual* distinction between a device and its instructions was made in the late 1930s and early 1940s with the invention of the universal, general purpose, programmable computer: a computer could now (in



(a) Tape reader of the ASCC. **(b)** Ruth Lichterman wiring the ENIAC.

Figure 1. Machine instructions for mechanical and electronic mainframe computers.

principle) be adapted to calculate anything, rather than just those specified in its original design. For mechanical and electromechanical mainframe computers (e.g., the Automatic Sequence Controller Calculator at Harvard University), these instructions were fed to the machine through spools of punched hole paper tapes (fig. 1a). In fully electronic mainframe computers (e.g., the Electronic Numerical Integrator And Computer), they were provided by connecting plugs to sockets, flipping switches, and turning dials (fig. 1b). However, in both cases the instructions were still firmly rooted in the physical, as using the machine for another "general purpose" still required rewiring or new paper tapes.

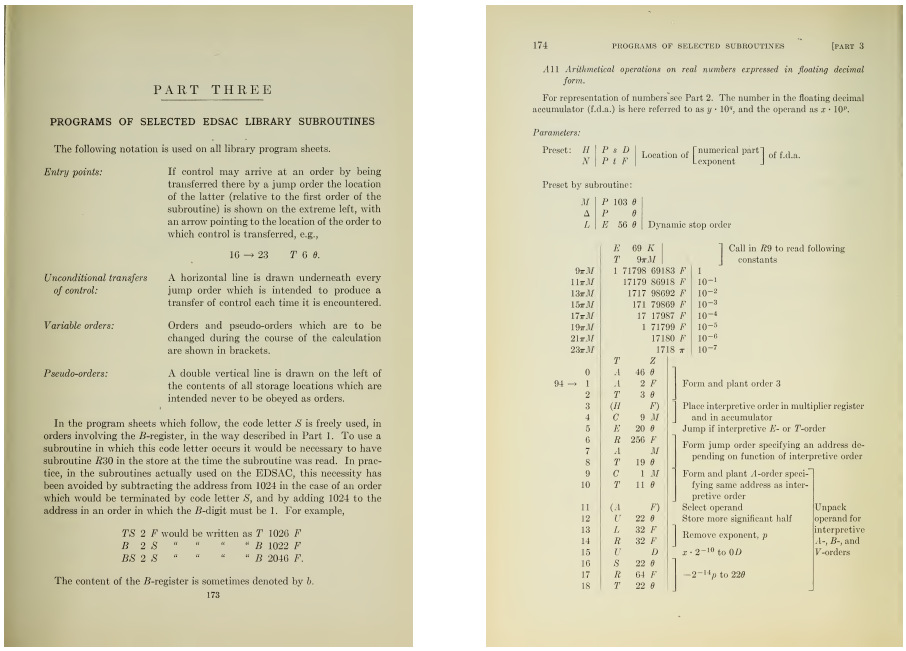
The first *material* distinction between the computer and its instructions was created in 1948-1949 through the successful implementation of stored-program architectures,[1] which allowed calculations to be executed from an electronic memory. For the first time, the device's instructions were purely electronic signals, and the physical state of the computer no longer represented the program it was calculating. To get the instructions into the electronic memory still required feeding them to the machine through punched paper cards or magnetic tapes, however, so this material separation was at least partly theoretical rather than experiential.

Software for these mainframe computers was written by the researchers working on the technology. In the first textbook on programming published in 1951 – "The Preparation of Programs for an Electronic Digital Computer" by Wilkes, Wheeler, and Gill; often referred to as just WWG[2] – more than half of the first edition of the book (85 out of 164 pages) contained the software developed by

[1] Nick Metropolis and Jack Worlton (1980). 'A Trilogy on Errors in the History of Computing.' In: *Annals of the History of Computing* 2.1, pp. 49–59.

[2] Martin Campbell-Kelly (2011). 'In praise of 'Wilkes, Wheeler, and Gill'.' In: *Communications of the ACM* 54.9, pp. 25–27.

the authors and their team working with the Electronic Delay Storage Automatic Calculator (EDSAC) (fig. 2).



PART THREE

PROGRAMS OF SELECTED EDSAC LIBRARY SUBROUTINES

The following notation is used on all library program sheets.

Entry points: If control may arrive at an order by being transferred there by a jump order the location of the latter (relative to the first order of the subroutine) is shown on the extreme left, with an arrow pointing to the location of the order to which control is transferred, e.g.,

$16 \rightarrow 23 \quad T\ 6\ \theta.$

Unconditional transfers of control: A horizontal line is drawn underneath every jump order which is intended to produce a transfer of control each time it is encountered.

Variable orders: Orders and pseudo-orders which are to be changed during the course of the calculation are shown in brackets.

Pseudo-orders: A double vertical line is drawn on the left of the contents of all storage locations which are intended never to be obeyed as orders.

In the program sheets which follow, the code letter S is freely used, in orders involving the B-register, in the way described in Part 1. To use a subroutine in which this code letter occurs it would be necessary to have subroutine K50 in the store at the time the subroutine was read. In practice, in the subroutines actually used on the EDSAC, this necessity has been avoided by subtracting the address from 1024 in the case of an order which would be terminated by code letter S, and by adding 1024 to the address in an order in which the B-digit must be 1. For example,

$TS\ 2\ F$ would be written as $T\ 1026\ F$
$B\ \ 2\ S$ " " " " $B\ 1022\ F$
$BS\ 2\ S$ " " " " $B\ 2046\ F$.

The content of the B-register is sometimes denoted by b.

173

174 PROGRAMS OF SELECTED SUBROUTINES [PART 3

A11 Arithmetical operations on real numbers expressed in floating decimal form.

For representation of numbers see Part 2. The number in the floating decimal accumulator (f.d.a.) is here referred to as $y \cdot 10^q$, and the operand as $x \cdot 10^p$.

Parameters:

Preset: H | $P\ s\ D$ | Location of [numerical part / exponent] of f.d.a.
N | $P\ t\ F$ |

Preset by subroutine:

M | P 103 θ |
Δ | P θ |
L | E 56 θ | Dynamic stop order

	E	69 K		Call in R9 to read following constants
	T	$9\pi M$		
$9\pi M$	1	71798	69183 F	1
$11\pi M$		17179	88918 F	10^{-1}
$13\pi M$		1717	88092 F	10^{-2}
$15\pi M$		171	79869 F	10^{-3}
$17\pi M$		17	17987 F	10^{-4}
$19\pi M$		1	71799 F	10^{-5}
$21\pi M$			17180 F	10^{-6}
$23\pi M$			1718 π	10^{-7}
	T		Z	
0	A	46 θ		
94 → 1	A	2 F		Form and plant order 3
2	T	3 θ		
3	$(H$		$F)$	Place interpretive order in multiplier register
4	C	9 M		and in accumulator
5	E	20 θ		Jump if interpretive E- or T-order
6	R	256 F		Form jump order specifying an address de-
7	A	M		pending on function of interpretive order
8	T	19 θ		
9	C	1 M		Form and plant A-order speci-
10	T	11 θ		fying same address as inter-
				pretive order
11	$(A$		$F)$	Select operand
12	U	22 θ		Store more significant half
13	L	32 F		Remove exponent, p
14	R	32 F		
15	U		D	$x \cdot 2^{-10}$ to $0D$
16	S	22 θ		
17	R	64 F		$-2^{-14}p$ to 22θ
18	T	22 θ		

Unpack operand for interpretive A-, B-, and V-orders

Figure 2. The first two pages of the program library included in Wilkes, Wheeler, and Gill's *The Preparation of Programs for an Electronic Digital Computer*.

They did not use the word software to describe their code, but instead referred to each self-contained calculation as a subroutine, and a collection of subroutines as a program. Considering the huge effort required to write correct code, the authors advised writing subroutines such that they could be reused, and described how they organised them into a code library. Interestingly, the publication of the book boosted the popularity of the EDSAC hardware design. Because of the lack of computer standardisation, software and hardware were still tightly coupled. To take advantage of the considerable labour represented by the WWG software library, other computers matched the EDSAC (e.g., Japan's first electronic computer, the Tokyo Automatic Computer[3]).

5.3 THE COOPERATIVE DESIGN OF SOFTWARE

While the mainframe computers of the 1940s were research prototypes or military equipment, the 1950s saw the rise of the computer as a commercial product. For example, IBM, with decades of success selling tabulating machinery to clients such as the the US Government and the Third Reich[4], announced its first foray into the commercial computing market in April 1952: the IBM 701 Electronic

[3] Campbell-Kelly, 'In praise of 'Wilkes, Wheeler, and Gill',' p. 2.

[4] From the start, computers have been inextricably linked to warfare, population control, and crimes against humanity.

Figure 3. The IBM 701 Electronic Data Processing Machine.

Data Processing Machine (fig. 3). While the hardware industry developed, the program code required to use the computer remained uncommodified, and still largely the prerogative of the computer user to develop. When the first eighteen customers received the IBM 701[5], it did not come with any software except for a primitive assembler and some "utilities" such as a one-card, memory clearing program.[6]

Almost from the beginning, the skills and time required to develop software have been the main constraints determining the design of the technology and the direction of its industry. The labour cost of programming the early mainframes was so high (often upwards of a year's rental of the device) that IBM considered it a major threat to the future commercial success of computers: organisations were reluctant to buy a new computer and have to redevelop their entire library of programs. With the launch of the IBM 704 on the horizon, an IBM sales manager in the Los Angeles area decided to organise a meeting between all customers of the 701 to discuss whether a collaborative software development effort would be possible to reduce the individual burden. This approach was a profound shift from the normal, internal way software was developed at the time, and no small undertaking, as it included companies that were each other's direct competitors (e.g., RAND Corporation, Lockheed Aircraft Corporation, North American Aviation Inc.), with a history of poaching each other's programmers. After several rounds of drinks, the November 1952 meeting would later be known as the suc-

[5] The first client to receive the 701 was the Los Alamos Scientific Lab in New Mexico, working on the development of nuclear weapons.

[6] Martin Campbell-Kelly (2004). *From airline reservations to Sonic the Hedgehog: a history of the software industry*. MIT press, p. 30.

cessful inauguration of the "Digital Computer Association" (DCA), colloquially known as the Drunkard's Computer Association.[7]

The DCA's efforts resulted in the first cooperative design of a piece of software for the 701 called, appropriately, PACT: the Project for the Advancement of Coding Techniques. Although the compiler was an impressive accomplishment and a better piece of software than anything offered by IBM, the DCA was convinced that "[t]he spirit of cooperation between member organizations and their representatives during the formulating of PACT-I has been one of the most valuable resources to come from the project. It is essential that this spirit of cooperation continue with future project plans".[8]

And continue it did. When IBM announced the 704 in May 1954, the groups involved in PACT started discussing a new joint effort to standardise assemblers and avoid the redundancy of each programming the same set of mathematical subroutines (e.g., numerical inversions) and utility programs (e.g., input-output programs). This eventually culminated in the founding of SHARE in 1955, the very first "User's organization" of the computing industry. Its explicit goals were "1) the standardization of machine language and certain machine practices, 2) the elimination of redundant effort expended in connection with use of the computer, 3) the promotion of inter-installation communication and 4) the development of a meaningful stream of information between the user and the manufacturer".[9] Within two years, the SHARE network of IBM 704 installations grew from eighteen to forty-seven. A Policy Committee was appointed to direct the efforts and distribute programming assignments to members. Once finished, these were made available to all other members as part of the SHARE distributions, disseminated via post by the SHARE Program Library Agency as magnetic tapes or punched cards (if not exceeding more than 1000 cards).[10]

The activities of the DCA and the organisations that emerged out of it offer a window to a fundamentally different mode of software production. The choice of the names PACT and SHARE both strongly signal commitments to plurality, cooperation, and compromise. A point of pride among the founders of SHARE was the strong spirit of collaboration between the representatives of highly competitive companies.[11] In each case, the defining principles of the networks were not just technical, but codified norms about what it meant to be part of the cooperative. As stated in the "Obligations of a SHARE Member" included in the SHARE Reference Manual for the IBM 704, "[t]he principle obligation of a member *is to have a cooperative spirit*"(emphasis original).[12] The obligations also included the

[7] Mary Brandel (1999). '1955: IBM customers form the first computer user group.' In: *CNN*. URL: http://edition.cnn.com/TECH/computing/9905/05/1955.idg/.

[8] Paul Armer (1956). 'SHARE - A Eulogy to Cooperative Effort.' In: *Annals of the History of Computing 2*.

[9] Steve Guendert (2011). 'Mainframe History and the First User's Groups (SHARE).' In: p. 2. URL: https://www.cmg.org/wp-content/uploads/2011/05/m_79_5.pdf.

[10] SHARE Program Library Agency (1977). 'User's Guide and Catalog of Programs.' In: URL: http://www.bitsavers.org/pdf/ibm/share/SHARE_PgmCatalog_Jan77.pdf.

[11] Atsushi Akera (2001). 'Voluntarism and the Fruits of Collaboration: The IBM User Group, Share.' In: *Technology and Culture* 42.4, 710–736. ISSN: 0040-165X.

[12] Joanna Edson et al. (1956). 'SHARE Reference Manual for the IBM 704.' In: URL: https://latimesblogs.latimes.com/thedailymirror/files/share59.pdf.

imperative that majorities should not be "overbearing" to minorities; they should try to win them over through discussion rather than simply vote them down.

Cooperative users' groups became a popular model for distributed software development around other computers as well: the UNIVAC 1100 had USE, IBM's 4300 had GUIDE, Wang Equipment had SWAP, etc. The user groups were not just about exchanging knowledge and software between members, but often had a formal relationship with computer manufacturers, and they were able to exert considerable power over the business decisions these made (e.g., IBM on multiple occasion tried to kill the VM/CMS Operating System but SHARE intervened on behalf of its users[13]).

5.4 SOFTWARE AS A PACKAGE

Up until the end of the 1950s, software was generally considered an artefact without inherent commercial value, and there was little interest in claiming control or ownership over it. The prevailing idea at the time was that software by definition was inimical to standardisation and could never be sold as is: a general ledger or payroll software system would always need to be built specifically for the locale it would be used in.[14] Rather, manufacturers treated software as a way to boost the viability of their hardware, and they budgeted its development as part of their marketing costs. They freely shared the software they or their customers built through mail-order program libraries, recurring newsletter catalogues from which members could request a punched card or magnetic tape copy for a small fee. These software were considered general blueprints and "seldom if ever used as programs; rather, the overall system design was usually modified and then recoded to the particular user's requirements".[15]

The rising number of computer installations across the United States and the rapidly expanding capabilities of computers created a strong demand for more software, yet the shortage of workers with programming skills and the slow speed of program development made it difficult to meet. The inability of the in-house developed, custom-built model of software development to meet these demands resulted in the emergence of the software contracting industry. Not only did this industry increase the overall quantity of software in the US, but their business models – different from the manufacturer- or user-originated code – also created a new way of imagining and designing software in the 1960s: the software package.

For the organisations that could not find or afford in-house programmers, hiring development services from a contractor was an attractive way to create the programs they needed. Initially, these contractors would develop custom software based on the specifications worked out with their clients. Their business models were based on economies of scope: they specialised in building software for a very narrow market so they could reduce development costs by building up

[13] Ed Scannell (1984). 'IBM User Groups.' In: *Computerworld* 8.October.

[14] Arthur Norberg (1983). 'An Interview with Walter Bauer.' In: *Charles Babbage Institute*. URL: https://conservancy.umn.edu/bitstream/handle/11299/107108/oh061wb.pdf.

[15] Larry Welke (1980). 'The Origins of Software.' In: *Datamation* December, p. 127.

knowledge about the domain and a library of program code that could be used as templates.[16] Eventually, this repurposing of previously built software by contractors evolved into a growing interest for "pre-packaged" software across the computer industry. What the concept of the package aspired to do, was "make a program portable, transferable, and usable on a second computer unrelated to the first",[17] a paradigmatic shift in the way software was imagined[18]. Rather than serve as just a blueprint for building a new system, software was beginning to be considered as something that could be reused across locales.

The '62 CFO (Consolidated Functions Ordinary) package developed by IBM for the 1401 and released through their Application Program Library was an early attempt at such a package for life insurance companies, used for billing and accounting. The intent was that it could be "used broadly throughout the Life Insurance Industry as operational computer programs or as a guide in the development of personalized total systems on an individual company basis".[19] It was developed in consultation with representatives from several life insurance and, by 1964, used relatively widely (peaking at 200-300 installations), resulting in the creation of three CFO user groups by the insurance companies.

However, the market for software as a package never really took off, and at the end of 1960 was only about 10 percent the size of contract programming. There were a couple reasons for this stagnation. First, the belief that custom-developed software was inherently superior to pre-built solutions remained strong. Second, the lack of hardware standardisation made software developed for one computer generally incompatible for others, so the pool of potential customers for a package was small. And third, the bundling of software together with hardware by computer manufacturers meant third-party package developers were competing against the manufacturers' free software.

5.5 THE RISE OF THE SOFTWARE PRODUCT

A number of events in the mid to late 1960s made the market for software as a package more viable, and eventually culminated in the emergence of software as a product.

The problem of hardware standardisation was addressed by IBM's 1964 announcement of their System/360: a family of mainframes that would all be compatible with each other, making vertical and horizontal integration across market segments possible. For software contractors with a backlog of packages, this sig-

[16] Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry*.
[17] Welke, 'The Origins of Software.'
[18] The meaning of the term *software package* was not as clear-cut as presented here. For some, it referred to pre-made software developed by a manufacturer whose costs were bundled together with the hardware and available through their software library. Others use it to refer to just pre-made software in general, regardless of origin or delivery. At least one collection of oral histories reports that it was called a package because the client received, as a package, the program code, documentation, and some level of service (e.g., installation, maintenance) (Luanne Johnson [2002]. 'Creating the software industry-recollections of software company founders of the 1960s.' In: *IEEE Annals of the History of Computing* 24.1, pp. 14–42)
[19] JoAnne Yates (1995). 'Application Software for Insurance in the 1960s and Early 1970s.' In: *Business and Economic History*, pp. 123–134.

nificantly increased the pool of potential customers for each piece of software and thus its potential profitability.

The problem of manufacturers bundling their software with the hardware was similarly up to IBM to address, although other factors played a role. The company Applied Data Research (ADR) had released a package in 1966 called Autoflow, which produced automatic flowcharts of Assembly-based software. It had sold 300 copies by the end of 1968, which was relatively successful, but ADR believed their sales had been severely hurt by the fact that IBM had started to offer a free package called Flowcharter, which also produced software flowcharts (although not automatically and not based on analyzing the assembly code). ADR eventually brought a lawsuit against IBM, claiming that it was monopolising the software industry by bundling software for free with its devices. Whether the ADR lawsuit, the simultaneous anti-trust case brought by the US government over IBM's market dominance, or just the rising costs of software development and support that was "virtually bringing the company to its knees",[20] IBM announced that starting 1 January 1970, it would "unbundle" its software from its hardware and start charging separately for it.[21]

This unbundling did not necessarily create a market overnight by removing the competition of free software (many early packages did not have a direct IBM equivalent), but it did force companies to consider software costs when buying computers, allowing the notion of non-bundled software as a reasonable alternative to emerge. Before unbundling, buying software as a stand-alone package was an unusual decision that needed to be justified, often by the computer controller or the president of the company. Once IBM had normalised the idea, purchasing authority (for products up to a certain price range, e.g., US$15.000-20.000) moved down in the organisational hierarchy, making it easier to sell software packages.[22]

The idea that custom-built software was superior to packages never really went away, but the economic crunch of the 1970s made hiring programming contractors or maintaining an in-house development team simply too expensive for many companies. Packages, despite its design limitations, were significantly cheaper and could be operational within a matter of weeks instead of months, generating a faster return on investment. Rather than not use any software at all, organisations chose to buy a pre-made package and instead redesign their company structure to fit its design.[23]

With these three barriers removed, the software industry expanded significantly. In 1970 the annual turnover for all US American software firms was less

[20] Johnson, 'Creating the software industry-recollections of software company founders of the 1960s,' p. 37.

[21] Thomas Haigh (2002). 'Software in the 1960s as Concept, Service, and Product.' In: *IEEE Annals of the History of Computing* 24.1, pp. 5–13.

[22] Johnson, 'Creating the software industry-recollections of software company founders of the 1960s,' p. 27.

[23] Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry*, p. 98.

than US$0.5 billion; by 1979 this had quadrupled to roughly US$2 billion.[24] Software magazines were created to showcase all the different packages available and conference papers were written about how to evaluate and choose the best ones.

Slowly, the software package turned into a capital good: a software product. This commodification of software left a lasting imprint on its dominant designs. An early representative example of this transformation was Informatic's Mark IV, a file management system that generated programs for creating, maintaining, and reading data from punched cards, magnetic tapes, and direct access devices. First released in 1967, it grew out of tailor-made packages, before being turned into a stand-alone "product". John Postley, inventor of the previous Mark packages, argued that turning the system into a successful product required it to be bug free, robust, and fully supported by stand-alone manuals and documentation. This might seem trivial, but was a significant shift from the much more forgiving requirements when delivering software as a package of code and customer service. The "free" software package provided by the computer manufacturer "was supplied with few contractual obligations, and no matter how well the customer might be supported in practice, there was no legal requirement that the manufacturer supply a fully operational application".[25] The software *product*, on the other hand, was "a discrete software artifact that required little or no customization, either by the vendor or by the buyer; it was actively marketed, it was sold or leased to a computer user, and the vendor was contractually obligated to provide training, documentation, and after-sale service".[26]

With the business model change from software as a 'free' package to software as a tradeable product, there was a shift in responsibility over the quality of that software, and a subsequent appropriation of control over the software by its producers in order to better manage those responsibilities and ensure continued profit. Previously, software code was easy to access and trivial to replicate, which was economically acceptable because it had little to no value outside of the specific context it was built for. Software contractors were hired more for their development and support services than the program. Distribution and replication of software did not seriously endanger their business models. Now that more and more commercial value was contained in the program code, and it was possible to run it on multiple machines without requiring customisation, protecting the software code became imperative for the revenue streams of the company.

To protect an informational asset, legal instruments had to be invented or reappropriated. Patents were an ill-fit because they were designed for tangible objects rather than immaterial goods. Copyright law required a human-readable copy of the artefact to be submitted to the Copyright Office, something companies were obviously reluctant to do. In the case of Mark IV, Informatics decided to use the mechanism of trade secret laws, and required employees and customers to sign non-disclosure agreements. Most importantly, however, after roughly thirty years of mostly decentralised ownership of programs, the commodification of software

[24] Martin Campbell-Kelly (1995). 'Development and structure of the international software industry, 1950-1990.' In: *Business and economic history*, pp. 73–110, p. 74.
[25] Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry*, p. 99.
[26] Ibid., p. 99.

led developers to implement the first intentional technological barrier to a software's negotiability: they shipped their program in binary code, with the express purpose to limit the ability of their customers to access, read, and customise the software.

5.6 THE FIRST WAVE OF MICROCOMPUTERS

The microcomputer industry developed largely independently from the mainframe and minicomputer sector, but the evolution of the industry and the design of software follows an almost identical path. The first wave of microcomputers were sold as assemble-yourself kits for electronics hobbyists, containing circuit boards, wires, card connectors, a case, and instructions of often questionable quality. The first commercially successful microcomputer was the Altair 8800, announced in the now-iconic January 1975 edition of Popular Electronics (fig. 4). This was not a device with much practical applicability out of the box, but a new toy that provided an interesting challenge for hardware homebrewers and software hackers, captured perfectly in the headline of a later ad campaign: "Building your own computer won't be a piece of cake. (But, we'll make it a rewarding experience)".[27] The microcomputer was programmed by flipping switches, and the output came as flashing LED lights on the front. Using it for anything more substantial than that required buying external peripherals such as a keyboard, screen, disk drive, RAM extensions -– products which the company behind the Altair, MITS, sold to make a profit on the microcomputer.

Analogous to the early mainframes, these microcomputers did not come with any operating system or software programs, so users had to design their own. Initially using mostly machine language, until programing language interpreters were released that allowed users to write in something more human-friendly. Famously, the Altair 8800 gave Bill Gates and Paul Allen their first opportunity and first product, the BASIC interpreter called Altair BASIC. Unheard of at the time, Gates and Allen were paid for their software in royalties and received a small percentage of each sale made by MITS, rather than the more common licensing lump-sum. This business model would provide them with a steady influx of capital over the next years as they diversified into other programming languages (COBOL, FORTRAN) and other hardware.

Digital Research, founded by the married couple Gary Kildall and Dorth McEwen, was the first vendor of operating systems. Between 1974 and 1976, Gary Kildall designed CP/M – Control Program for Microcomputers – which could be run with minimal changes on any machine using the 8008 Intel microprocessor. At a time when a manufacturer could expect to spend US$50,000-100,000 on developing an operating system for a microcomputer, Kildall sold his CP/M by mail-order for US$75, and eventually licensed it (non-exclusively) for US$25,000 to IMSAI, the manufacturer of the Altair rival IMSAI 8080. Hugely popular, Digital Research adapted CP/M to run on other microprocessors as well, and by the end of the

[27] n.a. (1975). 'Building your own computer won't be a piece of cake.' In: *Popular Electronics* 4.

Figure 4. The Altair 8800 on the cover of the January 1975 edition of Popular Electronics magazine.

Figure 5. The welcome screen and interface of the Electric Pencil word-processing application created by Michael Shrayer in 1976.

decade could run on 200 different computers.[28] In much the same way that IBM's System/360 increased the market for software packages, CP/M's ad-hoc standardisation expanded the possible market for personal software: anything adapted to that operating system could run on a multitude of devices, liberating developers from having to significantly rewrite the software for each platform.

The application program market was still barebones in the mid-1970s, constrained by the limitations of the hardware and the fledgling state of programming languages and operating systems. One of the few commercially successful "practical" software products for the microcomputers at the time was a word processor. Michael Shrayer developed *Electric Pencil* out of personal necessity when he had to write the documentation for an assembler package he had created. State-of-the-art at the time, it showed the message "THE ELECTRIC PENCIL (C) 1977 MICHAEL SHRAYER" when the program was launched (fig. 5); nothing else would happen until the user started typing, which would replace the text.[29] Originally released in December 1976 for the Altair 8800, it become a runaway success and Shrayer quickly found himself rewriting the application for more than seventy other devices.

[28] Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry*, p. 206.
[29] Michael Shrayer (1977). *The Electric Pencil Word Processor: Operator's Manual*, p. 6.

The majority of programs available at the time were text-based games. One of the first and most widely used ones,[30] Star Trek, demonstrates the collective and negotiated character of these early programs. Originally written for a Sigma 7 minicomputer between 1971 and 1972 by a teenager called Mike Mayfield, Star Trek was a text-based strategy game inspired by the popular *Spacewar!* game. During the summer of 1972, a local Hewlett-Packard sales office allowed Mayfield access to their computer, and he rewrote it for their version of BASIC. It was subsequently added to the HP Contributed Program Library in February 1973 under the name STTR1. Other members of the library reprogrammed the game for their own hardware architectures, and added new game features along the way. Eventually, some of the many versions of the Star Trek game made it to the microcomputers. A copy of the game for the Altair 8800, written by Lynn Cochran, was featured as the cover story of the June 1976 edition of the SCCS INTERFACE magazine. The article included the entire software code (on just two and a half pages) and a block diagram explaining the general logic of the program (fig. 6). The renegotiability and shared ownership of the software was an explicit part of its identity. One article in the magazine lists the many different versions of the Star Trek game, and how one could embark on the "Enterprise" of modifying their own copy.[31] An ad in the magazine for yet another copy of the game highlighted that, in addition to "a complete program source listing", the package would also include "tips on how to 'patch' the program to add your own features".[32]

This collaborative and open design of software (games, at least) was the dominant imaginary, and the notion that programs should not be commodified held considerable sway among hobbyists. Such ideas clashed with those who wanted to extract value from their software. Writing angrily in February 1976, Gates accused members of the Homebrew Computer Club: "As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid? Is this fair?".[33]

5.7 THE GOLD RUSH OF APPLICATION PROGRAMS

The second wave of microcomputers started in 1977, kicked off by devices such as the Commodore PET (January), Apple II (April), and TRS-80 (August). These machines approached what we now recognise as a (personal) computer, at least in its hardware design: it hid away the electrical wiring behind a molded plastic case and came with a keyboard built in (fig. 7).

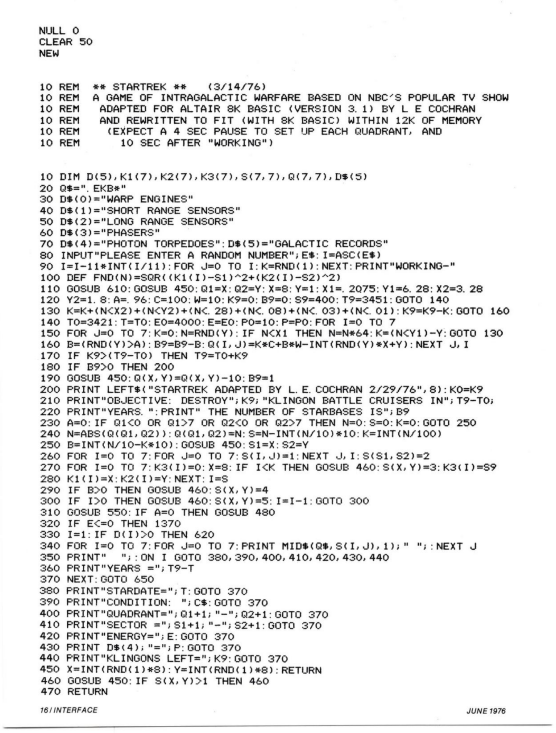
Although its exterior looked more approachable, significant technical skills were still required to overcome the lack of programs, poor documentation, and

[30] Art Childs (1976). 'Interfacial.' In: *SCCS INTERFACE* June. URL: https://archive.org/details/sccs_v1n7/, p. 2.

[31] Ralph Kiestadt (1976). 'Large Scale Systems: Software Choices for Star Trek.' In: *SCCS INTERFACE* June. URL: https://archive.org/details/sccs_v1n7/, p. 49.

[32] Inc. International Data Systems (1976). 'Patchable Star Trek/Space War Program Offered.' In: *SCCS INTERFACE* June. URL: https://archive.org/details/sccs_v1n7/, p. 41.

[33] Bill Gates et al. (1976). 'An open letter to hobbyists.' In: *Homebrew Computer Club Newsletter* 2.1, p. 2.



```
NULL 0
CLEAR 50
NEW


10 REM  ** STARTREK **   (3/14/76)
10 REM  A GAME OF INTRAGALACTIC WARFARE BASED ON NBC'S POPULAR TV SHOW
10 REM    ADAPTED FOR ALTAIR 8K BASIC (VERSION 3.1) BY L.E COCHRAN
10 REM    AND REWRITTEN TO FIT (WITH 8K BASIC) WITHIN 12K OF MEMORY
10 REM     (EXPECT A 4 SEC PAUSE TO SET UP EACH QUADRANT, AND
10 REM       10 SEC AFTER "WORKING")


10 DIM D(5),K1(7),K2(7),K3(7),S(7,7),Q(7,7),D$(5)
20 Q$=".EKB*"
30 D$(0)="WARP ENGINES"
40 D$(1)="SHORT RANGE SENSORS"
50 D$(2)="LONG RANGE SENSORS"
60 D$(3)="PHASERS"
70 D$(4)="PHOTON TORPEDOES":D$(5)="GALACTIC RECORDS"
80 INPUT"PLEASE ENTER A RANDOM NUMBER";E$:I=ASC(E$)
90 I=I-11*INT(I/11):FOR J=0 TO I:K=RND(1):NEXT:PRINT"WORKING-"
100 DEF FND(N)=SQR((K1(I)-S1)^2+(K2(I)-S2)^2)
110 GOSUB 610:GOSUB 450:Q1=X:Q2=Y:X=8:Y=1:X1=.2075:Y1=6.28:X2=3.28
120 Y2=1.8:A=.96:C=100:W=10:K9=0:B9=0:S9=400:T9=3451:GOTO 140
130 K=K+(NCX2)+(NCY2)+(NC.28)+(NC.08)+(NC.03)+(NC.01):K9=K9-K:GOTO 160
140 T0=3421:T=T0:E0=4000:E=E0:P0=10:P=P0:FOR I=0 TO 7
150 FOR J=0 TO 7:K=0:N=RND(Y):IF NCX1 THEN N=N*64:K=(NCY1)-Y:GOTO 130
160 B=(RND(Y)>A):B9=B9-B:Q(I,J)=K*C+B*W-INT(RND(Y)*X+Y):NEXT J,I
170 IF K9>(T9-T0) THEN T9=T0+K9
180 IF B9>0 THEN 200
190 GOSUB 450:Q(X,Y)=Q(X,Y)-10:B9=1
200 PRINT LEFT$("STARTREK ADAPTED BY L.E.COCHRAN 2/29/76",8):K0=K9
210 PRINT"OBJECTIVE:  DESTROY";K9;"KLINGON BATTLE CRUISERS IN";T9-T0;
220 PRINT"YEARS.":PRINT" THE NUMBER OF STARBASES IS";B9
230 A=0:IF Q1<0 OR Q1>7 OR Q2<0 OR Q2>7 THEN N=0:S=0:K=0:GOTO 250
240 N=ABS(Q(Q1,Q2)):Q(Q1,Q2)=N:S=N-INT(N/10)*10:K=INT(N/100)
250 B=INT(N/10-K*10):GOSUB 450:S1=X:S2=Y
260 FOR I=0 TO 7:FOR J=0 TO 7:S(I,J)=1:NEXT J,I:S(S1,S2)=2
270 FOR I=0 TO 7:K3(I)=0:X=8:IF I<K THEN GOSUB 460:S(X,Y)=3:K3(I)=S9
280 K1(I)=X:K2(I)=Y:NEXT:I=S
290 IF B>0 THEN GOSUB 460:S(X,Y)=4
300 IF I>0 THEN GOSUB 460:S(X,Y)=5:I=I-1:GOTO 300
310 GOSUB 550:IF A=0 THEN GOSUB 480
320 IF E<=0 THEN 1370
330 I=1:IF D(I)>0 THEN 620
340 FOR I=0 TO 7:FOR J=0 TO 7:PRINT MID$(Q$,S(I,J),1);" ";:NEXT J
350 PRINT"  ";:ON I GOTO 380,390,400,410,420,430,440
360 PRINT"YEARS =";T9-T
370 NEXT:GOTO 650
380 PRINT"STARDATE=";T:GOTO 370
390 PRINT"CONDITION:  ";C$:GOTO 370
400 PRINT"QUADRANT=";Q1+1;"-";Q2+1:GOTO 370
410 PRINT"SECTOR =";S1+1;"-";S2+1:GOTO 370
420 PRINT"ENERGY=";E:GOTO 370
430 PRINT D$(4);"=";P:GOTO 370
440 PRINT"KLINGONS LEFT=";K9:GOTO 370
450 X=INT(RND(1)*8):Y=INT(RND(1)*8):RETURN
460 GOSUB 450:IF S(X,Y)>1 THEN 460
470 RETURN
```



```
480 IF K<1 THEN RETURN
490 IF C$="DOCKED" THEN PRINT"STARBASE PROTECTS ENTERPRISE":RETURN
500 FOR I=0 TO 7:IF K3(I)<=0 THEN NEXT:RETURN
510 H=K3(I)*.4*RND(1):K3(I)=K3(I)-H:H=H/(FND(0)^.4):E=E-H
520 E$="ENTERPRISE FROM":N=E:GOSUB 530:NEXT:RETURN
530 PRINT H;"UNIT HIT ON ";E$;" SECTOR";K1(I)+1;"-";K2(I)+1;
540 PRINT" (";N;"LEFT)":RETURN
550 FOR I=S1-1 TO S1+1:FOR J=S2-1 TO S2+1
560 IF I<0 OR I>7 OR J<0 OR J>7 THEN 580
570 IF S(I,J)=4 THEN C$="DOCKED":E=E0:P=P0:GOSUB 610:RETURN
580 NEXT J,I:IF K>0 THEN C$="RED":RETURN
590 IF E<E0*.1 THEN C$="YELLOW":RETURN
600 C$="GREEN":RETURN
610 FOR N=0 TO 5:D(N)=0:NEXT:RETURN
620 PRINT D$(I);" DAMAGED.";
630 PRINT" ";D(I);"YEARS ESTIMATED FOR REPAIR.":PRINT
640 IF A=1 THEN RETURN
650 INPUT"COMMAND";A
660 IF A<1 OR A>6 THEN 680
670 ON A GOTO 710,310,1250,1140,690,1300
680 FOR I=0 TO 5:PRINT I+1;"= ";D$(I):NEXT:GOTO 650
690 IF D(4)>0 THEN PRINT"SPACE CRUD BLOCKING TUBES.";:I=4:GOTO 630
700 N=15:IF P<1 THEN PRINT"NO TORPEDOES LEFT":GOTO 650
710 IF A=5 THEN PRINT"TORPEDO ";
720 INPUT"COURSE (1-8.9)";C:IF C<1 THEN 650
730 IF C>=9 THEN 710
740 IF A=5 THEN P=P-1:PRINT"TRACK:";:GOTO 900
750 INPUT"WARP (0-12)";W:IF W<=0 OR W>12 THEN 710
760 IF W<.2 OR D(0)<=0 THEN 780
770 I=0:PRINT D$(I);" DAMAGED, MAX IS .2 ";:GOSUB 630:GOTO 750
780 GOSUB 480:IF E<=0 THEN 1370
790 IF RND(1)>.25 THEN 870
800 X=INT(RND(1)*6):IF RND(1)>.5 THEN 830
810 D(X)=D(X)+INT(6-RND(1)*5):PRINT"**SPACE STORM, ";
820 PRINT D$(X);" DAMAGED**":I=X:GOSUB 630:D(X)=D(X)+1:GOTO 870
830 FOR I=X TO 5:IF D(I)>0 THEN 860
840 NEXT
850 FOR I=0 TO X:IF D(I)<=0 THEN NEXT:GOTO 870
860 D(I)=.5:PRINT"**SPOCK USED A NEW REPAIR TECHNIQUE**"
870 FOR I=0 TO 5:IF D(I)=0 THEN 890
880 D(I)=D(I)-1:IF D(I)<=0 THEN D(I)=0:PRINT D$(I);" ARE FIXED!"
890 NEXT:N=INT(W*8):E=E-N-N+.5:T=T+1:S(S1,S2)=1
900 Y1=S1+.5:X1=S2+.5:IF T>T9 THEN 1370
910 Y=(C-1)*.785398:X=COS(Y):Y=-SIN(Y)
920 FOR I=1 TO N:Y1=Y1+Y:X1=X1+X:Y2=INT(Y1):X2=INT(X1)
930 IF X2<0 OR X2>7 OR Y2<0 OR Y2>7 THEN 1110
940 IF A=5 THEN PRINT Y2+1;"-";X2+1,
950 IF S(Y2,X2)=1 THEN NEXT:GOTO 1060
960 PRINT:IF A=1 THEN PRINT"BLOCKED BY ";
970 ON S(Y2,X2)-3 GOTO 1040,1020
980 PRINT"KLINGON";:IF A=1 THEN 1050
990 FOR I=0 TO 7:IF Y2<>K1(I) THEN 1010
1000 IF X2=K2(I) THEN K3(I)=0
1010 NEXT:K=K-1:K9=K9-1:GOTO 1070
1020 PRINT"STAR";:IF A=5 THEN S=S-1:GOTO 1070
1030 GOTO 1050:2L29E76C
1040 PRINT"STARBASE";:IF A=5 THEN B=2:GOTO 1070
1050 PRINT" AT SECTOR";Y2+1;"-";X2+1:Y2=INT(Y1-Y):X2=INT(X1-X)
1060 S1=Y2:S2=X2:S(S1,S2)=2:A=2:GOTO 310
1070 PRINT" DESTROYED!";:IF B=2 THEN B=0:PRINT".  .  GOOD WORK!";
```



```
1080 PRINT:S(Y2,X2)=1:Q(Q1,Q2)=K*100+B*10+S:IF K9<1 THEN 1400
1090 GOSUB 480:IF E<=0 THEN 1370
1100 GOSUB 550:GOTO 650
1110 IF A=5 THEN PRINT"MISSED!":GOTO 1090
1120 Q1=INT(Q1+W*Y+(S1+.5)/8):Q2=INT(Q2+W*X+(S2+.5)/8)
1130 Q1=Q1-(Q1<0)+(Q1>7):Q2=Q2-(Q2<0)+(Q2>7):GOTO 230
1140 I=3:IF D(I)>0 THEN 620
1150 INPUT"PHASERS READY:  ENERGY UNITS TO FIRE";X:IF X<=0 THEN 650
1160 IF X>E THEN PRINT"ONLY GOT";E:GOTO 1150
1170 E=E-X:Y=K:FOR I=0 TO 7:IF K3(I)<=0 THEN 1230
1180 H=X/(Y*(FND(0)^.4)):K3(I)=K3(I)-H
1190 E$="KLINGON AT":N=K3(I):GOSUB 530
1200 IF K3(I)>0 THEN 1230
1210 PRINT"**KLINGON DESTROYED**"
1220 K=K-1:K9=K9-1:S(K1(I),K2(I))=1:Q(Q1,Q2)=Q(Q1,Q2)-100
1230 NEXT:IF K9<1 THEN 1400
1240 GOTO 1090
1250 I=2:IF D(I)>0 THEN 620
1260 PRINT D$(I);" FOR QUADRANT";Q1+1;"-";Q2+1
1270 FOR I=Q1-1 TO Q1+1:FOR J=Q2-1 TO Q2+1:PRINT"   ";
1280 IF I<0 OR I>7 OR J<0 OR J>7 THEN PRINT"***";:GOTO 1350
1290 Q(I,J)=ABS(Q(I,J)):GOTO 1340
1300 I=5:IF D(I)>0 THEN 620
1310 PRINT"CUMULATIVE GALACTIC MAP FOR STARDATE";T
1320 FOR I=0 TO 7:FOR J=0 TO 7:PRINT"  ";
1330 IF Q(I,J)<0 THEN PRINT"***";:GOTO 1350
1340 E$=STR$(Q(I,J)):E$="00"+MID$(E$,2):PRINT RIGHT$(E$,3);
1350 NEXT J:PRINT:NEXT I:GOTO 650
1360 PRINT:PRINT"IT IS STARDATE";T:RETURN
1370 GOSUB 1360:PRINT"THANKS TO YOUR BUNGLING, THE FEDERATION WILL BE"
1380 PRINT"CONQUERED BY THE REMAINING";K9;"KLINGON CRUISERS!"
1390 PRINT"YOU ARE DEMOTED TO CABIN BOY!":GOTO 1430
1400 GOSUB 1360:PRINT"THE FEDERATION HAS BEEN SAVED!"
1410 PRINT"YOU ARE PROMOTED TO ADMIRAL":PRINT K0;"KLINGONS IN";
1420 PRINT T-T0;"YEARS.  RATING=";INT(K0/(T-T0)*1000)
1430 INPUT"TRY AGAIN";E$:IF LEFT$(E$,1)="Y" THEN 110
```

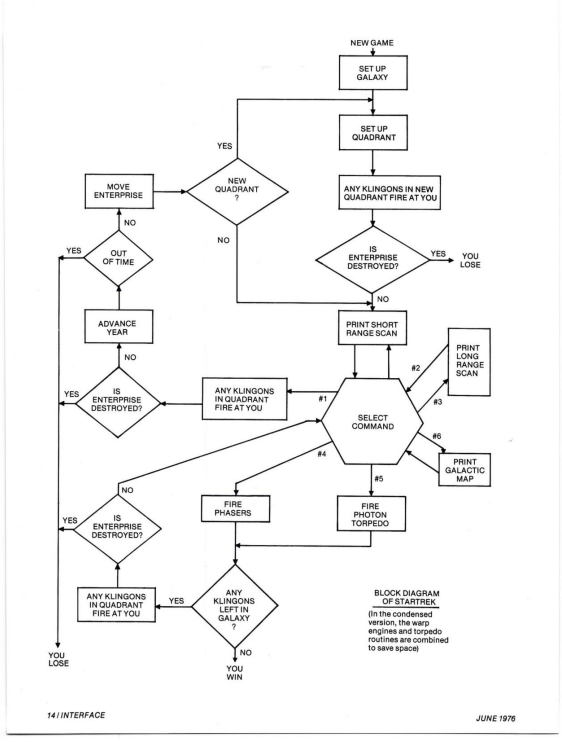



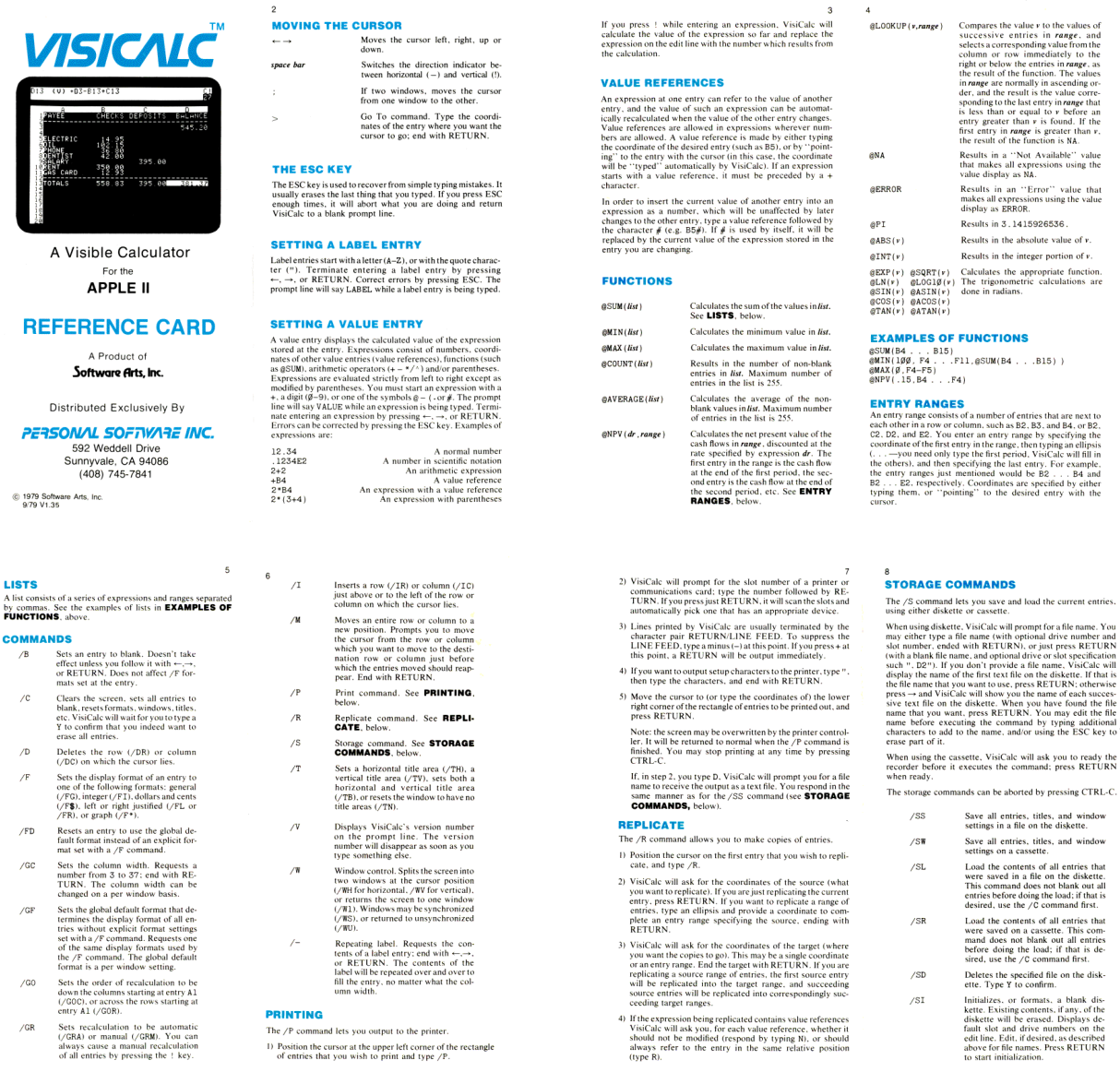
Figure 6. The program code and logic flowchart for the Star Trek game written by Lynn Cochran, published in the June 1976 edition of the SCCS INTERFACE magazine.

Figure 7. The second wave of commercial microcomputers. From left to right, the Commodore PET, Apple II, and TRS-80.

minimal support so the computer could be used as a tool rather than a toy. This changed at the end of the 70s, when the success of VisiCalc helped establish the application software market and usher in a four year gold rush that set the stage for the design and industry of software for years. Released initially for the Apple II in 1979 (and significantly buoying the platform's sales), VisiCalc was a spreadsheet program that would dynamically recalculate cell values in response to changes in other cells. Compared to other software available at the time, VisiCalc was exceptionally easy to use – an explicit design choice strongly influenced by the fact that its developers – Dan Bricklin and Bob Frankston – conceived of VisiCalc as "a product, not a program".[34] Whereas mainframe software relied on a low volume of sales for high prices (US$5,000 - US$250,000), the developers of VisiCalc decided to bet on a high volume-low price mass-market strategy instead, targeting everyday individuals instead of just large corporations with deep pockets. Supporting thousands of customers would be expensive, if not impossible, so Dan Bricklin and Bob Frankston relied on its design and documentation to be so simple and unsurprising that they did not have to have any further interaction with the customer after the point of sale: any feature that could not be concisely explained on the reference card was dropped (fig. 8). Priced initially at US$99, the company sold 1,000 copies per month on average. For four straight years, from 1979 through 1983, Personal Software/VisiCorp was the largest software company on the market.

Although games continued to be one of the main types of software, spreadsheets and similar "practical" applications such as word processors, database managers, and business graphics became multimillion dollar categories in their own rights. The professional appeal of the microcomputer was further solidified when IBM announced it would enter the business as well, and released the IBM PC in 1981. Big Blue's reliable reputation helped it quickly capture a large market share, but more important for the software industry was that the IBM PC had an

[34] Robert M. Frankston (2015). 'Implementing VisiCalc.' In: URL: https://rmf.vc/implementingvisicalc.



VISICALC™

A Visible Calculator

For the

APPLE II

REFERENCE CARD

A Product of

Software Arts, Inc.

Distributed Exclusively By

PERSONAL SOFTWARE INC.

592 Weddell Drive
Sunnyvale, CA 94086
(408) 745-7841

2

MOVING THE CURSOR

←→	Moves the cursor left, right, up or down.
space bar	Switches the direction indicator between horizontal (–) and vertical (!).
:	If two windows, moves the cursor from one window to the other.
>	Go To command. Type the coordinates of the entry where you want the cursor to go; end with RETURN.

THE ESC KEY

The ESC key is used to recover from simple typing mistakes. It usually erases the last thing that you typed. If you press ESC enough times, it will abort what you are doing and return VisiCalc to a blank prompt line.

SETTING A LABEL ENTRY

Label entries start with a letter (A–Z), or with the quote character ("). Terminate entering a label entry by pressing ←, →, or RETURN. Correct errors by pressing ESC. The prompt line will say LABEL while a label entry is being typed.

SETTING A VALUE ENTRY

A value entry displays the calculated value of the expression stored at the entry. Expressions consist of numbers, coordinates of other value entries (value references), functions (such as @SUM), arithmetic operators (+ – * / ^) and/or parentheses. Expressions are evaluated strictly from left to right except as modified by parentheses. You must start an expression with a +, a digit (0–9), or one of the symbols @ – (. or #. The prompt line will say VALUE while an expression is being typed. Terminate entering an expression by pressing ←, →, or RETURN. Errors can be corrected by pressing the ESC key. Examples of expressions are:

12.34	A normal number
.1234E2	A number in scientific notation
2+2	An arithmetic expression
+B4	A value reference
2*B4	An expression with a value reference
2*(3+4)	An expression with parentheses

3

If you press ! while entering an expression, VisiCalc will calculate the value of the expression so far and replace the expression on the edit line with the number which results from the calculation.

VALUE REFERENCES

An expression at one entry can refer to the value of another entry, and the value of such an expression can be automatically recalculated when the value of the other entry changes. Value references are allowed in expressions wherever numbers are allowed. A value reference is made by either typing the coordinate of the desired entry (such as B5), or by "pointing" to the entry with the cursor (in this case, the coordinate will be "typed" automatically by VisiCalc). If an expression starts with a value reference, it must be preceded by a + character.

In order to insert the current value of another entry into an expression as a number, which will be unaffected by later changes to the other entry, type a value reference followed by the character # (e.g. B5#). If # is used by itself, it will be replaced by the current value of the expression stored in the entry you are changing.

FUNCTIONS

@SUM(*list*)	Calculates the sum of the values in *list*. See **LISTS**, below.
@MIN(*list*)	Calculates the minimum value in *list*.
@MAX(*list*)	Calculates the maximum value in *list*.
@COUNT(*list*)	Results in the number of non-blank entries in *list*. Maximum number of entries in the list is 255.
@AVERAGE(*list*)	Calculates the average of the non-blank values in *list*. Maximum number of entries in the list is 255.
@NPV(*dr*,*range*)	Calculates the net present value of the cash flows in *range*, discounted at the rate specified by expression *dr*. The first entry in the range is the cash flow at the end of the first period, the second entry is the cash flow at the end of the second period, etc. See **ENTRY RANGES**, below.

4

@LOOKUP(*v*,*range*)	Compares the value *v* to the values of successive entries in *range*, and selects a corresponding value from the column or row immediately to the right or below the entries in *range*, as the result of the function. The values in *range* are normally in ascending order, and the result is the value corresponding to the last entry in *range* that is less than or equal to *v* before an entry greater than *v* is found. If the first entry in *range* is greater than *v*, the result of the function is NA.
@NA	Results in a "Not Available" value that makes all expressions using the value display as NA.
@ERROR	Results in an "Error" value that makes all expressions using the value display as ERROR.
@PI	Results in 3.1415926536.
@ABS(*v*)	Results in the absolute value of *v*.
@INT(*v*)	Results in the integer portion of *v*.
@EXP(*v*) @SQRT(*v*) @LN(*v*) @LOG10(*v*) @SIN(*v*) @ASIN(*v*) @COS(*v*) @ACOS(*v*) @TAN(*v*) @ATAN(*v*)	Calculates the appropriate function. The trigonometric calculations are done in radians.

EXAMPLES OF FUNCTIONS

```
@SUM(B4 . . . B15)
@MIN(10@, F4 . . .F11,@SUM(B4 . . .B15) )
@MAX(@,F4-F5)
@NPV(.15,B4 . . .F4)
```

ENTRY RANGES

An entry range consists of a number of entries that are next to each other in a row or column, such as B2, B3, and B4, or B2, C2, D2, and E2. You enter an entry range by specifying the coordinate of the first entry in the range, then typing an ellipsis (. . .—you need only type the first period, VisiCalc will fill in the others), and then specifying the last entry. For example, the entry ranges just mentioned would be B2 . . . B4 and B2 . . . E2, respectively. Coordinates are specified by either typing them, or "pointing" to the desired entry with the cursor.

5

LISTS

A list consists of a series of expressions and ranges separated by commas. See the examples of lists in **EXAMPLES OF FUNCTIONS**, above.

COMMANDS

/B	Sets an entry to blank. Doesn't take effect unless you follow it with ←, →, or RETURN. Does not affect /F formats set at the entry.
/C	Clears the screen, sets all entries to blank, resets formats, windows, titles, etc. VisiCalc will wait for you to type a Y to confirm that you indeed want to erase all entries.
/D	Deletes the row (/DR) or column (/DC) on which the cursor lies.
/F	Sets the display format of an entry to one of the following formats: general (/FG), integer (/FI), dollars and cents (/F$), left or right justified (/FL or /FR), or graph (/F*).
/FD	Resets an entry to use the global default format instead of an explicit format set with a /F command.
/GC	Sets the column width. Requests a number from 3 to 37; end with RETURN. The column width can be changed on a per window basis.
/GF	Sets the global default format that determines the display format of all entries without explicit format settings set with a /F command. Requests one of the same display formats used by the /F command. The global default format is a per window setting.
/GO	Sets the order of recalculation to be down the columns starting at entry A1 (/GOC), or across the rows starting at entry A1 (/GOR).
/GR	Sets recalculation to be automatic (/GRA) or manual (/GRM). You can always cause a manual recalculation of all entries by pressing the ! key.

6

/I	Inserts a row (/IR) or column (/IC) just above or to the left of the row or column on which the cursor lies.
/M	Moves an entire row or column to a new position. Prompts you to move the cursor from the row or column which you want to move to the destination row or column just before which the entries moved should reappear. End with RETURN.
/P	Print command. See **PRINTING**, below.
/R	Replicate command. See **REPLICATE**, below.
/S	Storage command. See **STORAGE COMMANDS**, below.
/T	Sets a horizontal title area (/TH), a vertical title area (/TV), sets both a horizontal and vertical title area (/TB), or resets the window to have no title areas (/TN).
/V	Displays VisiCalc's version number on the prompt line. The version number will disappear as soon as you type something else.
/W	Window control. Splits the screen into two windows at the cursor position (/WH for horizontal, /WV for vertical), or returns the screen to one window (/WI). Windows may be synchronized (/WS), or returned to unsynchronized (/WU).
/–	Repeating label. Requests the contents of a label entry; end with ←, →, or RETURN. The contents of the label will be repeated over and over to fill the entry, no matter what the column width.

PRINTING

The /P command lets you output to the printer.

1) Position the cursor at the upper left corner of the rectangle of entries that you wish to print and type /P.

7

2) VisiCalc will prompt for the slot number of a printer or communications card; type the number followed by RETURN. If you press just RETURN, it will scan the slots and automatically pick one that has an appropriate device.

3) Lines printed by VisiCalc are usually terminated by the character pair RETURN/LINE FEED. To suppress the LINE FEED, type a minus (–) at this point. If you press + at this point, a RETURN will be output immediately.

4) If you want to output setup characters to the printer, type ", then type the characters, and end with RETURN.

5) Move the cursor to (or type the coordinates of) the lower right corner of the rectangle of entries to be printed out, and press RETURN.

Note: the screen may be overwritten by the printer controller. It will be returned to normal when the /P command is finished. You may stop printing at any time by pressing CTRL-C.

If, in step 2, you type D, VisiCalc will prompt you for a file name to receive the output as a text file. You respond in the same manner as for the /SS command (see **STORAGE COMMANDS,** below).

REPLICATE

The /R command allows you to make copies of entries.

1) Position the cursor on the first entry that you wish to replicate, and type /R.

2) VisiCalc will ask for the coordinates of the source (what you want to replicate). If you are just replicating the current entry, press RETURN. If you want to replicate a range of entries, type an ellipsis and provide a coordinate to complete an entry range specifying the source, ending with RETURN.

3) VisiCalc will ask for the coordinates of the target (where you want the copies to go). This may be a single coordinate or an entry range. End the target with RETURN. If you are replicating a source range of entries, the first source entry will be replicated into the target range, and succeeding source entries will be replicated into correspondingly succeeding target ranges.

4) If the expression being replicated contains value references VisiCalc will ask you, for each value reference, whether it should not be modified (respond by typing N), or should always refer to the entry in the same relative position (type R).

8

STORAGE COMMANDS

The /S command lets you save and load the current entries, using either diskette or cassette.

When using diskette, VisiCalc will prompt for a file name. You may either type a file name (with optional drive number and slot number, ended with RETURN), or just press RETURN (with a blank file name, and optional drive or slot specification such ",D2"). If you don't provide a file name, VisiCalc will display the name of the first text file on the diskette. If that is the file name that you want to use, press RETURN; otherwise press → and VisiCalc will show you the name of each successive text file on the diskette. When you have found the file name that you want, press RETURN. You may edit the file name before executing the command by typing additional characters to add to the name, and/or using the ESC key to erase part of it.

When using the cassette, VisiCalc will ask you to ready the recorder before it executes the command; press RETURN when ready.

The storage commands can be aborted by pressing CTRL-C.

/SS	Save all entries, titles, and window settings in a file on the diskette.
/SW	Save all entries, titles, and window settings on a cassette.
/SL	Load the contents of all entries that were saved in a file on the diskette. This command does not blank out all entries before doing the load; if that is desired, use the /C command first.
/SR	Load the contents of all entries that were saved on a cassette. This command does not blank out all entries before doing the load; if that is desired, use the /C command first.
/SD	Deletes the specified file on the diskette. Type Y to confirm.
/SI	Initializes, or formats, a blank diskette. Existing contents, if any, of the diskette will be erased. Displays default slot and drive numbers on the edit line. Edit, if desired, as described above for file names. Press RETURN to start initialization.

Figure 8. The VisiCalc reference card, minus the last page which showed an annotated screenshot of the interface. The reference card was shipped in a brown, fake leather folder together with 5 1/4" diskettes, a manual, and a registration card. See http://www.bricklin.com/history/saiproduct1.htm for more detail.

open hardware architecture (in contrast with the proprietary Apple II system, for example), making it possible for third-parties to develop software and hardware for the platform. The IBM hardware design became a de-facto standard, reducing (but not eliminating) the technical and business challenges of device-specific software development, and broadening the software market. The lack of practical software had kept the microcomputer a hobbyist novelty, but these applications and the legitimisation by IBM helped reimagine the microcomputer as an essential business tool for white-collar workers and a household economics device for non-technical users.

In addition to applications themselves, there also emerged a lively after-market software industry (referred to as add-ons, add-ins, plug-ins, accessories). Applications were often not just a tool for executing the pre-programmed functions, but also a development environment in their own right, or at the very least exposed enough of their code to make it possible for other code to interoperate with it. George Tate, CEO of Ahston-Tate, the company behind the immensely popular database application dBASE, explained how they "never really sold dBASE as just an end-user product. [...] It's always been – always and forever – a product that serves two markets". One was to use the database as a database, another was for skilled users to turn their workflow into a program "and have something that his secretary could use, or have something he could market".[35] The "accessories" built around dBASE fell into two categories: first, utilities to enhance the parent software (e.g., Quickcode by Fox&Geller helped with the creation of databases); second, "vertical market" packages, which added functionality to dBASE that served a particular niche community (e.g., Abstrat by Anderson-Bell added statistical analyses for accounting). Many of the other dominant software packages had and supported similar extensibility, allowing the software to be renegotiated by its users through third-party packages or programming their own customisations.

The particular way microcomputer applications were commodified reconfigured the means and relations of software production, and thus the potential for negotiating its design. Where software contractors and manufacturers relied on economies of scope, application programs for the microcomputer were based on economies of scale. This mass-marketisation intentionally disconnected the software's user from the software's developer, closing off one way people would traditionally customise the software. Because early consumers of applications were mostly technical hobbyists, the software's designs did allow for reprogramming, either by making the source code accessible or supporting the development of new functionality from within the application itself. However, because mass-marketisation also expanded the target market for software from niche homebrewers to everyday users, the proportion of consumers who had the skills (or interest) to reprogram their software themselves decreased significantly. Without the competences or access to more capable peers, the negotiation of a software's design was now limited to whatever add-ons and plugins were available to the user.

[35] Scott Mace (1983). 'Software accessories enhance software programs.' In: *InfoWorld* 5.10, p. 24.

5.8 THE SEARCH FOR SOFTWARE INTEGRATION

After the gold rush era of application software, the rest of the 1980s saw the software industry mature. By 1983, a handful of applications had established themselves as the market leaders and barriers to entry made it harder for small start-ups to compete. The increased technical capabilities of microcomputers meant software development became more complex and expensive, the human-computer interaction expertise necessary to build easy-to-use software was mostly locked inside existing companies, and an expensive advertising blitz was required to secure a limited spot on the shelf in the computer store (an estimated 35,000 products were competing for the 200 slots).[36]

In addition to these socioeconomic factors, there was also a simple technological bottleneck on the growth of the personal software market: it was practically impossible to use more than one application at a time. First, switching software would generally take several minutes, as the user had to close the current application, remove the storage disk, load the second disk, and start up the new program. Second, the lack of data standardisation meant that transferring data between application was generally impossible, unless you were lucky and the software application had made their format public and was popular enough that others built compatibility with it. Third, applications all had their own set of commands that had to be memorised. Understanding the basic functions of a piece of software could take weeks, and mastering all of them even longer. Having multiple applications use the same commands for different outcomes made it practically unrealistic for a user to learn more than two or three programs.

"Software integration" became the holy grail for applications in the 1980s. Its aim was to let the consumer use multiple programs side-by-side, easily share data between them, and unify the commands and interface so that skills would be transferable. This drive towards integration was the main impetus for most software innovation during the 1980s and helped application design converge into what we know today. There were three main designs the industry explored: application families, integrated packages, and windowed application managers.

5.8.1 *Application families*

Application families often started after a software developer achieved some success with one of the four main software products – word processors, spreadsheets, database managers, and graphing tools – and then tried to branch out to the other categories. These applications were purchased separately, and "integration" meant they had a shared command structure and data format. The main disadvantage of this "family" approach was that a user had to buy all the different applications to benefit from the integration, and while the company's spreadsheet might be the industry leader, there was no guarantee that their word processor would be better than (or ever comparable to) other applications on the market.

[36] Campbell-Kelly, 'Development and structure of the international software industry, 1950-1990,' p. 97.

Data format compatability was not reserved just for applications of the same family. Bob Frankston, one of the developers behind the VisiCalc application, created the text-based Data Interchange Format (DIF) so that VisiCalc could exchange data with VisiPlot, their graphing application. By virtue of VisiCalc's market dominance, DIF became an ad-hoc standard because other software companies tried to use read/write compatability with VisiCalc to claim a share of the market. This network effect also worked in favour of the dominant application, so companies would often build explicit compatability support into their data format. For example, the files of Ashton-Tate's dBASE II -– one of the best-selling database programs -– began with 500 characters listing the names, types, and sizes of all database fields, followed by the data itself, which made it possible for developers of other BASIC programs to build around it. WordStar, one of the largest word processor packages released in 1979, stored its data in a plain text file, making reading and linking to it even easier.

While these approaches made data integration technologically possible, experientially there was still a big disconnect between applications, because a user would still have to quit one program and load another to work with that data.

5.8.2 *Integrated packages*

Integrated application packages similarly tried to make it possible for customers to combine the most commonly used productivity software, but rather than selling these as separate applications, the idea was to build "one-application-to-rule-them-all". ContextMBA (1981) was the first of these integrated application packages, although it preceded the term. It combined a spreadsheet, database manager, word processor, form creator, graphics tool, and "telecommunications" functionality (i.e., using phone lines to transfer data between the application and a mainframes computer, other ContextMBA users, or a timeshare system). Each of these were referred to as a "context" and a user could create live links between data entered in any of them, so if a spreadsheet was updated, so were the graphs created based on that data.

Since there was still no standardised microcomputer architecture, ContextMBA was written in Pascal, a higher-level language that could be run on any device with a Pascal interpreter, but which also made it slower than if it was written for a specific architecutre in assembly or machine language. It proved too slow for most users. Reviews in PC Magazine of June 1983 summarised that ContextMBA "sacrifices efficiency for integration"[37] and concluded that "[t]he idea of an integrated management system is excellent. [...] Nevertheless, a good set of compatible text editing, database, graphics, and spreadsheet programs that runs under MS-DOS is probably a better investment".[38]

Lotus 1-2-3 was the second integrated package that was released, and turned out to be an overnight success (fig. 9). Launched in October 1982 and shipped in January 1983, Lotus 1-2-3 was the brainchild of Mitch Kapor, former head of

[37] Mark S. Zachmann (1983). 'Context MBA: Half a Step in the Right Direction.' In: *PC Magazine* 2.1, p. 123.
[38] Ibid., p. 131.

development at VisiCorp and creator of the popular VisiPlot and VisiTrend accessory applications for VisiCalc. The software was more limited than ContextMBA and arguably only barely qualifies as an 'integrated' package: it was mostly a spreadsheet program with additional database, word processing, and graphics capabilities. It was integrated in the sense that all these components were usable in the spreadsheet and shared the same base commands, but they were not fully-fledged applications. Data from VisiCalc, dBase II, and WordStar could also be added after being "analysed" by Lotus.

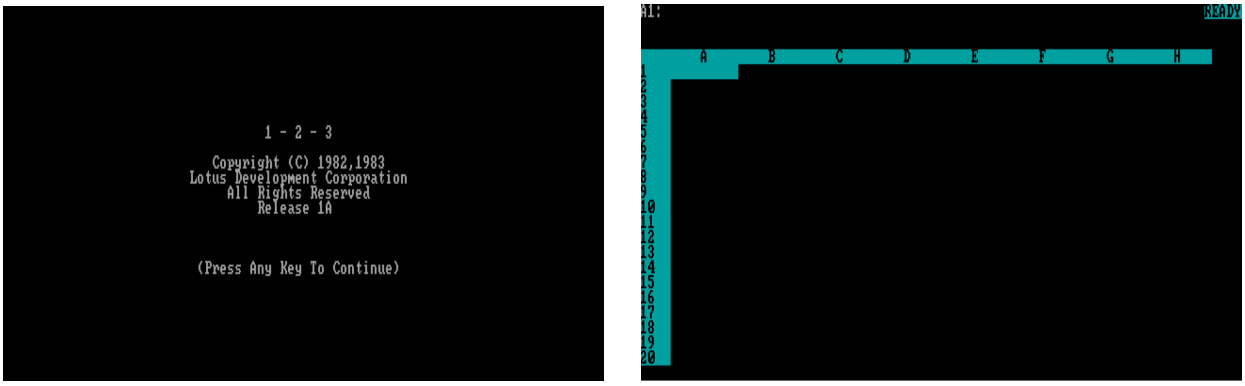


Figure 9. The splash screen and spreadsheet view of Lotus 1-2-3 release 1a (1983).[39]

One of Lotus' most popular features was that it allowed users to write macros – sequences of keystroke commands that could automate part of the user's interaction with the software and system and thus create quite complex programs (e.g., parsing comma-delimited files). This generated a very active after-market industry of macro packages (in v1.0) and add-ins (in v2.0) (fig. 10). These were initially seen as parasitic by Lotus, but were embraced when it became clear that the grassroots development of additional functionality boosted its popularity in niche communities and helped fortify its market position. It needed this, because Lotus never successfully diversified its offerings. Nearly all of its other software products were commercial failures.

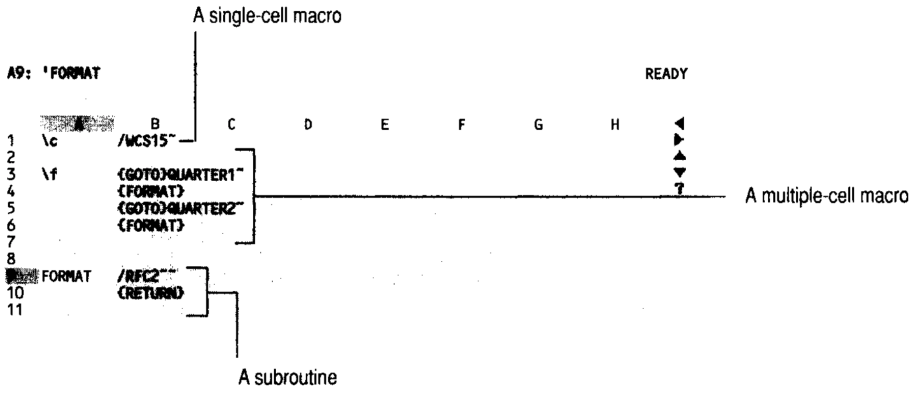


Figure 10. An example macro of Lotus 1-2-3 release 2.3 (1991).[40]

When version 2.0 of Lotus was released in September 1985, little new was added beyond supporting add-on development by exposing how other software could integrated with Lotus. Lotus Corporation also created considerable institutional support for add-on developers. They published all the file formats of its products, set up the Development Services Department to provide telephone support, organised conferences (the first one was attended by 400 people), and created the Registered Developer Program. Before these changes, insider knowledge or reverse engineering of Lotus was required to build an add-on, but with explicit documentation and support, the numbers of third-party software exploded. In 1987, Lotus magazine reported that over 1.000 add-ons were created.

Although Lotus' extensive support for add-on development made its design explicitly negotiable, the company fought heavily to retain control in other ways, with ramifications for negotiable designs in the software industry at large. Competitors to Lotus often tried to synchronise their menu tree structure with the one from Lotus. This was helpful to users because they did not have to learn how to use a new interface, but it also made it possible for the macros that they or their organisation had written – which could reach thousands of commands and represented a significant investment – to be interoperable with the other software's menu as well. Lotus Development Corp responded to these designs that would allow users to have more control over their software environment by filing copyright infringement lawsuits. Their first victim was Paperback Software International in 1987, which had developed a spreadsheet that looked identical to Lotus 1-2-3. At the time, the legal system only recognised source code as something that could be copyrighted, so it required a new interpretation of copyright to also include what became known as the "look and feel" of a piece of software. This lawsuit was a widely publicised event because it was believed that it could reshape the vision of what the software industry was working towards. One attorney observed in the magazine InfoWorld that "[i]f Lotus wins these lawsuits, it ... may kill all hope of there someday being a standard user interface across all software packages".[41]

Lotus Development Corp won the lawsuit in 1990, and it quickly went on to also sue Borland International, developer of the competitor application Quattro Pro. Quattro Pro had its own unique interface, but also a mode that looked the same as Lotus 1-2-3, and thus allowed macros developed in one to work in the other. The lawsuit dragged on for six years, with multiple conflicting verdicts at different levels in the US court structure, before eventually reaching the Supreme Court where it ended in a 4-4 stalemate between the judges. By default, this meant an earlier judgement in favour of Borland would stand, and menu hierarchies were deemed uncopyrightable. However, the six years of uncertainty for developers had had a chilling effect on technical interoperability and visual synchronicity as something that should be pursued in software design (in particular because the first verdict against Borland happened in Massachusetts, where a lot of software developers had their offices). Without these qualities, the freedom to easily switch between applications without having to abandon the skills and data developed in

[41] Doug Derwin (1987). 'Overheard...' In: *InfoWorld* 9.4, p. 35.

Figure 11. Lotus' Symphony (left) and Ashton-Tate's Framework. Note the explicit revocation of user ownership and control in Framework's splash screen: "You do not become the owner of the package nor do you have the right to copy or alter the software".

another was significantly constrained, and instead users were becoming walled-off as soon as they made their first choice.

The integrated packages industry reached its peak in 1984 with the simultaneous launch of Lotus Development Corps' Symphony and Ashton-Tate's Framework (fig. 11). Symphony was meant to be the full-blown integrated package that 1-2-3 aspired to be and included the classic combination of spreadsheet, word processor, graphics, database, and communications. Users could switch between each of these environments by pressing ALT+F10 and see a different view of the same underlying data. Ashton-Tate's Framework included the same applications, and organised them using the metaphor alluded to in its name. Each application was launched inside a frame, and the user could have multiple frames open at the same time, providing visual integration. Frames could contain other frames, data could be linked between frames such that changes would propagate, or frames could target other frames as output containers.

Integrated packages never delivered on their promises and were mostly commercial failures. The size of the software meant that almost all of them required an additional external hard disk to store the programs – or instead continuously swap between a fistful of floppy disks – and made them much slower than other applications. Even though the command structures were integrated, the sheer number of commands available made these software synonymous with difficult

to use in popular media.[42] The quality of, for example, the spreadsheet and the word processor part of the package could vary considerably, so while users paid a premium price for the integrated packaged they also have to buy other stand-alone applications. Data transfer also fell short of the claims made by vendors, with many integrated packages offering just cut-and-paste techniques for sharing data between different programs, rather than transclusion and live links.[43]

5.8.3 *Windowed Application Managers*

In addition to the multi-purpose application approach to integrated software, software developers also explored application managers that could show multiple, unrelated programs side-by-side and allow users to quickly switch between them: the window environment These efforts were not separate from the large integrated package approach, but happened in conjunction with them, before eventually being pitted against each other in the press.

The 1981 Xerox Star was the first commercial product that had a *graphical* window environment. The user accessed the software through icons and executed operations through menus. It also introduced the mouse as the main way to interact with those elements. This helped reduce the need for unified or standardised command structures that was one of the driving forces behind full package integration, because now users could see the different possible operations and click on them, rather than having to memorise the key combination that would execute it.

The 1983 Apple Lisa – on paper an acronym for Locally Integrated Software Architecture, but in reality named after the child Steve Jobs refused to acknowledge – is credited for popularising the graphical window manager (fig. 12). The Lisa was never a commercial success for a number of reasons, including its hefty price-tag, internal conflicts at Apple, and the fact that the more limited but cheaper Macintosh was released a year later. On the Lisa, the user could see multiple applications in the integrated package side-by-side and quickly switch between them (but not use them at the same time) through overlapping windows. This window environment was tied together with the integrated packaged called the Lisa Office System, which included LisaWrite, LisaCalc, LisaDraw, LisaGraph, LisaProject, LisaList, and LisaTerminal. To transfer data between these different applications, the Lisa used cut-copy-and-paste commands, and introduced the term 'clipboard' to refer to the temporary storage where the data would be kept in the process. Compared to the live links between data that other software offered, this model of data integration was considered to be on the most limited side of the spectrum.

Other dominant software companies quickly followed with their own windowed application managers: IBM's TopView, QuarterDeck's DESQView, Digital Research's Graphics Environment Manager, VisiCorp's VisiOn, and Microsoft's Windows (fig. 13). Some came with integrated packages, others were simply frameworks for third party software. By virtue of being the leading company for microcomputer

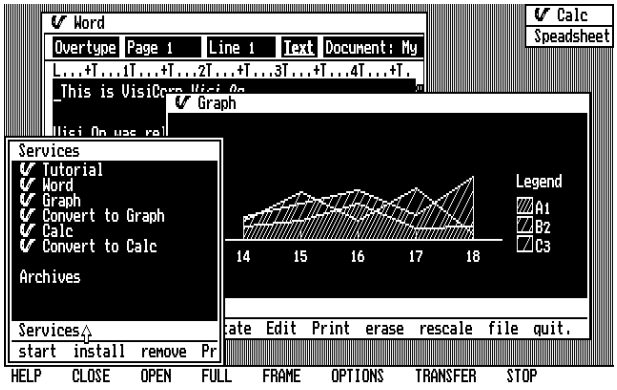
[42] Christine McGeever (1984). 'A Look at Lotus for the Mac.' In: *InfoWorld* 6.47.
[43] Paul Korzeniowksi (1984). 'Multi-application Packages: Who Needs Them?' In: *InfoWorld* 18.33.

Figure 12. Advertisement for the Apple Lisa in Personal Computing 1983.[44] Note the promotion of the Lisa Office programs and the cut and paste functionality.

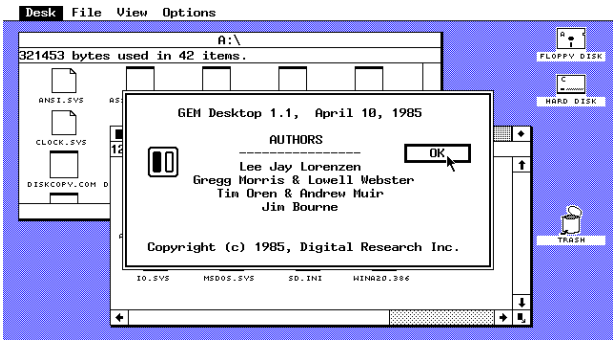
software in revenue, VisiCorp's product was the most highly anticipated. It turned out to be a fatal product for the company. VisiCorp had been working on their version since 1982 and committed significant resources, and when it was finally launched in January 1984 for the IBM PC it got positive reviews but made hardly any sales. The minimum hardware requirements were too high for most users and even a price cut from US$495 to US$95 within the first month did not save it. Another factor was that Microsoft had undercut VisiCorp by announcing in November 1983 that it would release their window-based environment the next spring. It chilled the interest for VisiOn – something better might be just around the corner – but Microsoft was overconfident and after months of technical difficulties only released their product – Windows 1.0 – in January 1985.

The "window wars" of 1984-1985 were the most exciting event in the software industry at the time. Integrated packages and windowed environments were two paths towards the same goal – functional multitasking — and were constantly pitted against each other in the press. Some industry observers proclaimed that the window environment would be the death of integrated software packages, while

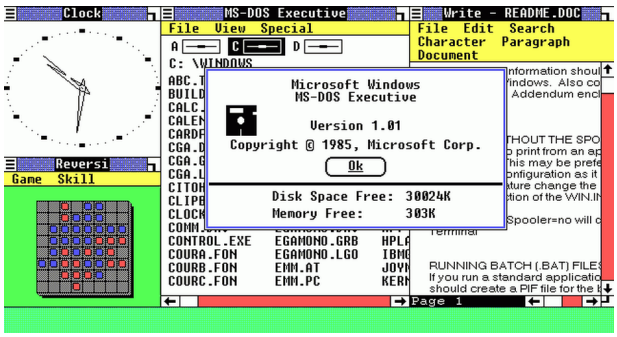
(a) IBM TopView (1985)



(b) VisiCorp VisiOn (1984)



(c) Digital Research Graphic Environment Manager (1985)



(d) Microsoft Windows (1985)

Figure 13. Windowed Application Managers released between 1984-1985.

others believed that they could co-exist.[45] Like the integrated packages, however, all early window environments were market failures. The software required too much from the hardware at the time and third-party applications needed to be rewritten from scratch to take advantage of the added benefits, which few were keen to do. The casualties were steep. VisiCorp and Digital Research had to fire half of its employees. Eventually, VisiCorp was sold off to Paladin Software, and Digital Research CEO Gary Kildall (of CP/M fame) resigned.[46] Only IBM and Microsoft had enough capital to overcome the financial blow of these failures.

5.8.4 *Component Software*

Component software was seen as the answer to the failures of integrated packages and windowed application managers: a hybrid model which would allow customers to use multiple applications from different vendors side-by-side, while still getting the data compatibility and functional interoperation from integrated packages. Writing about the battle between integrated software and windowing

[45] Winn L. Rosch (1985). 'Can Integrated Software Co-Exist with Windows?' In: *PC Magazine* 4.2, p. 60.

[46] Campbell-Kelly, *From airline reservations to Sonic the Hedgehog: a history of the software industry*, p. 250.

environments, one journalist predicted that "[s]oftware will become like stereo equipment – the low end will be integrated and the high end will be components".[47] In the late 1980s and early 1990s, it was touted as the next paradigm shift in computing. Apple confidently wrote: "In the 1980s, the graphical user interface revolutionized personal computing, enabling big leaps in user productivity and ultimately making obsolete all the applications standards of the day. In the 1990s, Apple believes the next major software revolution will be component software".[48] Compound documents would be the resulting artefact, "something like a display desktop that can contain visual and information objects of all kinds"[49] including text, graphics, spreadsheets, calendars, video, buttons, a newsreel, etc. The goal of this paradigm, stemming from the philosophy of object oriented programming, was to decenter the application as the container of commands and programs, and instead have the document take centre stage, with the data editable in place through contextual menus. For software developers, it was projected that they would move away from building fully-fledged applications and instead a lively component market would appear.[50]

Still developing and promoting its graphical window environment, Microsoft was an early proponent of this software vision and released the Dynamic Data Exchange (DDE) standard in 1987 as part of Windows 2.0, which "lets users dynamically link applications, automating tasks performed between multiple programs".[51] The first commercial software to demonstrate this ability was Microsoft Excel, also launched in 1987, but it took another two years for third-party applications to appear that took advantage of it. For example, Autocad could connect its diagrams to Microsoft Excel, such that changes in the spreadsheet would also update the graphics. Microsoft pushed these ideas further in 1990 and released their Object Linking and Embedding (OLE) protocol together with Windows 3.0, the first of their windowed application managers that was actually successful. Where DDE just propagated plain text messages between applications, OLE could maintain active links between data across programs and fully embed the content. This allowed an application to render a kind of content it was not able to normally create itself, and was specifically designed to support compound documents.

Apple, weighed down by its internal power struggle, finally followed with their own object linking model called Publish and Subscribe, released together with the Apple System 7 in 1991. The concepts were inspired by their successful application/development environment HyperCard and allowed content to be linked together across programs, both on a local computer and on a network. Users could publish documents on a shared folder, and other people on the network could subscribe to those documents and incorporate them in their own. Updates to the original data would automatically propagate to all other instances of the file that incorporated it. ClarisWorks, probably the most successful application on the Apple platform, was an integrated package that successfully implemented

[47] Rosch, 'Can Integrated Software Co-Exist with Windows?' p. 60.
[48] Apple Computer (1995). 'Macintosh vs. Windows 95: OpenDoc.' In: URL: http://tech-insider.org/mac/research/acrobat/Mac/950829.pdf.
[49] Hossein Bidgoli (2004). 'The internet encyclopedia (Volume 2).' In: p. 23.
[50] Laurie Flynn (1989). 'Applications for DDE are Starting to Appear.' In: *InfoWorld* 11.44, p. 13.
[51] Ibid., p. 13.

the component philosophy (if not the document-centric approach). Data types were contained in components and could be copied across the different document types, which meant that a text object could be added to a graphics document and the user would be able to access the word-processing capabilities of its parent application through it.

Lotus Development Corp., still limping along after the VisiOn fiasco, also developed an object-linking technology called Link, Embed, Launch-to-edit (LEL). Similar to Microsoft and Apple's technology, it allowed users to copy-paste objects between applications, have changes propagate through the maintained link, and launch the original application by clicking on the embedded object.[52]

Despite considerable investment in these software models by major players in the industry, the systems never quite seemed to materialise. PC Magazine, writing in 1994 about software suites, aptly captured the state these designed seemed to languish in for years: "Lotus, Microsoft, and WordPerfect have all made progress, but the seamlessly integrated suite still seems a version or two away".[53] Exactly why component software failed to become the new dominant design is unclear. Steve Jobs killed Apple's project when he retook control over the company, but its progress had already been sluggish for some time. Lotus Corporation was sold off to IBM in 1995 and never finished building the technology. Microsoft's OLE protocol is still alive and implemented in its Office suite, but third-party applications do not seem to take any advantage of it. Perhaps the protocols simply took too long to be established and the rest of the software market moved on from the idea. Perhaps, after years of competition, it was too difficult for these companies to collaborate and agree on a standard. Regardless, after fifteen years of the entire software industry working towards software integration, the dominant design that crystallised was a pale reflection of the initial goals. Visual integration, in the sense that multiple applications could be used side-by-side, was finally achieved when graphical application managers were accepted, thanks to Microsoft's persistence and long financial breath. But there was no standardisation of commands, menu structures, or interface designs that would allow users to easily transfer skills or programs between software. If there was file compatibility it was more likely to be a historical leftover than a planned feature, and there was virtually no functional interoperability at all, historic or otherwise. In the end, the walled-garden model of software won out, shedding reprogramming the code, writing macros, installing add-ons, or combining the functionality of multiple applications as the last few options users had to negotiate the design of applications.

5.9 CONCLUSION

The application model of software is a construct that was profoundly shaped by its progressive commodification between the 1950s and 1990s. It has resulted in

[52] Doug Barney (1993). 'Object Linking Readied for Unix Notes Clients, Programs.' In: *InfoWorld* 15.24, p. 18.
[53] Michael J. Miller (1994). 'Are They Suites Yet.' In: *PC Magazine* 13.18, p. 159.

a dominant design that centralises control with the software's developers while constraining its negotiability for other stakeholders.

In the early days of mainframe computers, software was not considered to have any inherent commercial value. Users had to write their own code and were free to share it with others. When computer manufacturers realised that software would become a bottleneck for the financial viability of their hardware industry, they invested company resources to support the creation of user-groups: communities of computer users who cooperatively designed their software, reducing development time by eliminating redundant efforts. During this time, software was a collectively negotiated artefact, and the only barriers to participate in that discussion were side-effects of the complexity of the technology.

As the hardware market matured and the share of computers with similar architectures increased, it became possible to reuse previously written code. This meant that it now had some value in and of itself, leading to emergence of the software contractor and software package. Rather than writing the code for the context in which it would be used, companies were now willing to purchase prebuilt software that was customised with the help of the contractor. While still negotiable, this process of software development (and the costs associated with it) made it more common for people to simply accept its design and adjust their operations, rather than the other way around.

While moderately profitable, developers of software packages wanted to increase the value they could extract from their software, so they leveraged the legal system against computers manufacturers who were still providing free software bundled with their other products, and worked towards normalising the idea that software was a product you had to pay for. This transformation further imbued software code with economic value, incentivising the developers to consolidate control over it by implementing the first intentional constraints to the software's negotiability. They obfuscated the source code – making it impossible for users to copy or adapt its design – and required their customers to sign agreements relinquishing full ownership.

In the mid 1970s, the microcomputer appeared as a new platform for a software industry to develop around. Early microcomputer games and applications were collective artefacts whose code was accessible and distributed through magazines for hobbyists to redesign. Once the microcomputer became affordable and approachable enough that general consumers could purchase one, it transformed software into a mass-market product. User-friendliness became an important way to successfully sell to non-technical users, abstracting away the complex insides of software as much as possible and relinquishing the developer from the responsibility to help the user understand and customise their program.

However, the limitations of the microcomputer hardware and the dominant design of the software throttled the growth of the application market, because people could only realistically use two or three programs at the same time. To resolve this bottleneck on their potential revenues, the entire software industry worked towards achieving "software integration". Applications competing for a spot on the user's device built in data transfer mechanisms, functional interoperability, and support for add-ons, all to make it easier for their software to be

used alongside others. Once on top, however, they were quick to use technical and legal means to limit the control others had over their software and thus market share, trampling user's abilities to renegotiate the application's design by reprogramming the source code, extending it through add-ons, or combining it with other applications. Once the dust had settled, users were left with a walled-garden design of software applications that could be used side-by-side visually, but had no interaction technologically.

With both the mainframe and microcomputer, we see the same process of progressive appropriation of control over the software by its developers, and the shrinking ability of others to (re)negotiate its design. Before commodification, when the users and uses of the computer were niche, the distribution of control over the technology was a side-effect of its complexity. As the commercial potential of the technology increased, and value was imbued in the software itself, profit-seeking companies redesigned the technology, law, and business practices to consolidate their autocratic power. More collective imaginations of software facilitated by openness, interoperability, and compatibility were supported when it broadened their market shares, but were aggressively shut down if it meant potentially losing customers and revenue to competitors. Today, software negotiation has devolved into consumer choice between not-so-different products, rather than actual personalisation of the technology itself. Computing technology that was developed under different political economies give us hints about what our computational media could have looked like – France's dirigisme produced the Minitel, Chile's socialism generated project Cybersyn. If we want to change the application model of software, those counterfactuals might serve as guiding lights.