5

Intangible Machines

'Software is or may be viewed as a new kind of property as, in effect, intangible machinery'.[1]

INTRODUCTION

In August 1967 Morton Jacobs, who was patent counsel for the Princeton-based software company Applied Data Research, wrote a letter to the Vice President and General Counsel of IBM, Burke Marshall, complaining about the way IBM was distributing its computer programs. Specifically, Jacobs complained that in giving customers who leased or bought IBM 360 computers a free copy of their Flowcharter program (which automatically developed flowcharts that set out the logic of a computer program) that IBM was destroying Applied Data Research's market for its Autoflow program (which also automatically developed flowcharts).[2] Jacobs added that in violation of antitrust laws and the law of unfair competition, IBM's actions would destroy his 'client's market for its Autoflow systems and destroy the property value of his clients software system by setting a "free" price as its market value, and make it impossible for our client to compete with IBM in the sale of such systems, all to the great detriment and injury of our client'.[3]

The main concern for Applied Data Research was that IBM distributed its Flowcharter program 'in the same way it distributed other software' whereby the software was 'given away "free" to IBM customers and "tied-in" to the sale of IBM/360

[1] Lawrence I. Boonin, 'Future Developments' as cited in C. McOustra, 'Legal Protection for Computer Programs' (1966) 8(4) *The Computer Journal* 289, 294.

[2] *Applied Data Research* v. *International Business Machines Corporation* 69 Civ, 1682 (filed 22 April 1969). Robert W. Wild, 'Computer Program Protection: The Need to Legislate a Solution' (1969) 54(4) *Cornell Law Review* 586, 588. ADR settled its antitrust suit with IBM for $2 million. Martin A. Goetz, 'How ADR Got Itself into the Software Products Business and Found Itself Competing against IBM' (1998) Computer History Museum 2.

[3] Letter written by Morton C. Jacobs to Mr. B. Marshall (16 August 1967), Charles Babbage Institute, Applied Data Research, Software Products Division records, CBI 154, Box 3, Folder 6.

computers'. Following a response from Marshall denying liability and saying that IBM would need to wait for Applied Data Research's patent (then pending) to issue on Autoflow, Jacobs responded saying that there 'can be no question that IBM's tie-in of its Flowcharter program to sales of the IBM 360 machines has misled potential customers of Applied Data Research's Autoflow into thinking that the IBM Flowcharter program is free'. On the basis that this 'illegal and anticompetitive practice by IBM has seriously injured [Applied Data Research's] sales', Jacob warned that unless steps were taken to correct these actions in violation of the antitrust laws Applied Data Research would be compelled to seek appropriate legal and enforcement remedies'.[4]

Applied Data Research followed up on its threat in April 1969 when it brought an antitrust action against IBM arguing that in giving its Flowcharter programs away for free that IBM had uncompetitively impeded the independent development of software.[5] Along with similar antitrust actions by Programmatics (a subsidiary of Applied Data Research), Control Data Corporation, and Data Processing Financial General Corporation, the Department of Justice also filed an antitrust action against IBM in 1969 arguing that by giving away software services for free and by bundling software with related equipment hardware under a single pricing plan (without detailing the price of the elements), IBM had engaged in anticompetitive practices that restrained actual or potential competitors from entering the relevant markets.[6]

In June 1969, IBM announced that from January 1, 1970, it would unbundle its software and hardware and charge separate prices for programming services and software packages. That is, it would separately price and sell the software it had previously bundled with its computers and sold at a single price. True to its word, in 1970 IBM not only began to sell hardware and software separately, it also began to charge a monthly fee for the use of its software or as IBM preferred, its 'program products'.

IBM's unbundling of hardware and software was part of a wide ranging set of changes that occurred in the computing industry in the 1960s and 1970s.[7] As the President of Programming Sciences Corporation, Albert M. Loring, said at the time, IBM's announcement 'has, in effect, given birth to the software industry as an Industry'.[8] As we will see, these changes also triggered a wide-ranging debate about the applicability of patent protection for the products of this emerging new industry.

[4] Letter written by Morton C. Jacobs to Burke Marshall (4 October 1967), Charles Babbage Institute, Applied Data Research, Software Products Division records, CBI 154, Box 3, Folder 6.

[5] *Applied Data Research* v. *International Business Machines Corporation* 69 Civ 1682 (SDNY 1969).

[6] *United States* v. *IBM* 69 Civ 200 (SDNY 1969). One of the complaints was that IBM had 'committed a fraud on the US Patent Office by applying for and obtaining patents for computer systems based on software but disguised as hardware'. Howard R. Popper, 'From Hardware to Software: An Adventure Having Some Surprises' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 120.

[7] Watts S. Humphrey, 'Software Unbundling: A Personal Perspective' (January–March 2002) *IEEE Annals of the History of Computing* 59, 62.

[8] Alan Drattel, 'Unbundling: The User Will Pay for the Works' (August 1969) *Business Automation* 36, 40.

At first blush, it may appear that IBM's decision to unbundle its (intangible) software from its (tangible) hardware represents a dematerialisation of patentable subject matter: a change from the situation where the subject matter consisted of a tangible machine that embodied intangible software to a situation where there were now two potential forms of subject matter – the tangible computer hardware and the intangible instructions used to control a computer (the program). Building upon the idea that the computer program represented a turning point in the 'long struggle for information to emancipate itself from the shackles of materiality',[9] there is a sense in which the unbundling of hardware and software marks yet another situation where patentable subject matter was dematerialised.

While IBM's decision to unbundle its software set in play a process that ultimately led to the dematerialisation of computer-related subject matter, this did not occur until the end of the twentieth century. This is because while in the 1970s software and hardware may have been unbundled from a commercial and marketing perspective, they remained technologically intertwined.[10] As we will see, following the decision to accept a technological reading of the subject matter, computer-related inventions in patent law retained a physical form. This remained the case for the remainder of the twentieth century.

To explore the role that materiality played in patent law's interaction with computer-related subject matter, how the law ultimately dealt with an unbundled dematerialised subject matter, and the role that computer science and the computer industry more generally played in these processes, this and the following two chapters explore patent law in the United States from the 1960s through to the early part of the twenty-first century. After looking at how software was created and consumed in the 1960s and as this changed how it gave rise to questions about the role intellectual property might play in the emerging software industry, I look at the contrasting ways that patentable subject matter was seen within the information technology industry more broadly. In Chapter 6, I turn to look at the problems patent law experienced in the 1960s and 1970s in attempting to reconcile the conflicting views within the industry about what the subject matter was and how it should be interpreted. In Chapter 7, I show how in the 1980s patent law came to view computer-related subject matter through the lens of 'abstractness' and the role that materiality played in determining the fate of that subject matter in this context. I also look at how as a result of changes in technology, patent law gradually shifted away from the materiality of the subject matter to look at its 'specificity' and how in so doing the subject matter was dematerialised.

[9] Jean-Francois Blanchette, *Burdens of Proof: Cryptographic Culture and Evidence Law in the Age of Electronic Documents* (Cambridge, MA: MIT Press, 2012), 18.

[10] One of the issues that patent law grappled with until the early 1980s was how to reconcile these contrasting ways of thinking about the subject matter: that is, how was patent law to reconcile the computer program as a commercially unbundled and independent object from the computer program which was technologically bound to the computer hardware (which mirrors the tension that arises because a patent is both a commercial and a technical document).

SOFTWARE IN THE 1960S

Today, software is typically thought of as a pre-packaged consumer product that contains the instructions or code that controls computers. As software historians have shown, this was not always the case. 'Historically speaking … software was not something that was purchased off-the-shelf, nor was it a single application or product. Rather it was a bundle of systems, services and support'.[11] Indeed, it was not until the late 1960s that software came to be treated as a product and 'even then software as code represented only a small component of a larger software system of services and support'.[12] While there were exceptions, in the 1960s there were no stand-alone companies specializing in the creation and sale of software products: there was no organized and discrete software industry to speak of, or at least as we understand it today.

In this environment, users tended to obtain their software in one of four ways. In some situations, corporate programming staff would write the software in-house. While manufacturers provided customer support and training, users often developed their own custom written programs. Another important source of software in the 1960s were the user groups that had been established to facilitate the sharing of programs, algorithms, and associated information. By 1960, around 20 different user groups exchanged programs for free.[13] Interestingly, the user groups were actively promoted and supported by the hardware manufacturers. For example, in order to alleviate the expense of programming that occurred when IBM replaced the 701 computer with the 704 model in 1954,[14] IBM formed SHARE (Society to Help Avoid Redundant Effort) to exchange programs amongst members (over 300 programs were shared) and to 'serve as a conduit between users and IBM's future development in hardware and programming'. The success of these early user groups encouraged the development of similar groups by IBM, other manufacturers, and industry bodies such as the American Bankers Association (who set up a 'Swap Room' at their annual conference to facilitate the exchange of computer programs).[15]

Another important source of software in the 1960s were the programming service (or custom programming) companies who produced custom written software for users on a fee basis. Typically, these software contractors wrote bespoke programs for corporate and government customers.[16] Often the programs were very expensive

[11] Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2010), 6.

[12] Ibid., 7.

[13] Robert F. Brothers and Alan M. Grimaldi, 'Prater and Patent Reform Proposals' (1969) 17 *Catholic University Law Review* 389, 392.

[14] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 33.

[15] Robert Head, 'The Travails of Software Resources' (January–March 2002) *IEEE Annals of the History of Computing* 82, 84–85.

[16] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 3–4.

($1 million not being uncommon[17]) and specifically written for particular organizations. As most of the sales occurred through personal contacts of staff or were in response to requests for custom software,[18] software contractors often had close working relationships with their customers.

The computer manufacturers who provided software free of charge to customers who bought hardware from them were another important source of software in the 1960s.[19] While the software packages were often costly to produce – figures spanned from a million dollars through to the $50 to $60 million IBM reportedly invested in its IBM/360 software – nonetheless, hardware manufacturers bundled the cost of the software into the cost of the hardware.[20] At the time, there was no thought of recovering the cost of developing software by selling or leasing it separately.[21] Instead, in an environment where programs were distributed freely as an inducement to purchase hardware, software development was often seen as a marketing cost. In other situations the marketing and sale of software were presented as the selling of services (which, it was hoped, would take any illegal tie-ins outside the scope of antitrust laws).[22] Either way, software was bundled with the hardware and given away for free as part of the overall package that was provided to customers.

The manner in which software was created, exchanged, and consumed during the 1960s had an impact on how software was valued.[23] It also had an impact on what was expected or demanded of the law. In relation to the software that was obtained for free from computer manufacturers or user groups, there was no need or interest in legal protection. To the extent that software was seen as a proprietary object, there was also little call for legal protection. This was because the close personal relationships that existed between software contractors and the companies they created software for minimized the need for legal or extra-legal means to control the reproduction or imitation of software. To the extent that there was a concern with

[17] Ibid.
[18] Ibid., 5.
[19] This was dominated by eight large companies: IBM, Honeywell Information Services, Univac, Burroughs Corporation, Control Data Corporation, National Cash Register Company, Digital Equipment Corporation, and Xerox.
[20] For example, under its single price or bundling procedures, IBM charged customers a single price based on the hardware supplied. Morton C. Jacobs, 'Computer Technology (Hardware and Software): Some Legal Implications for Antitrust, Copyright and Patents' (1970) 1 *Rutgers Journal of Computers and Law* 50, 62.
[21] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 98.
[22] Morton C. Jacobs, 'Computer Technology (Hardware and Software): Some Legal Implications for Antitrust, Copyright and Patents' (1970) *Rutgers Journal of Computers and Law* 50, 62. Anon, 'Software Gets a Hardsell Approach' (21 October 1967) *Business Week* 171.
[23] To the extent that software was obtained for free from computer manufacturers or through user groups, it helped to create the perception of software as 'objects without intrinsic value, or at best with value that there were no market mechanisms to realize'. Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 96.

software and how it circulated, the focus was on the body of the programmer rather that the product of their labour (the software). At a time when the major assets of software service companies would 'go down in the elevator every night',[24] the primary concern was the pirating of programmers (or 'body snatching'[25]) by competing firms and customers[26] rather than the copying or piracy of software. This was reflected in the fact that a key legal concern of the information technology industry at the time was the role that the law could play in restricting the mobility of workers via employment contracts, restraint of trade, non-compete contract clauses, and confidentiality agreements. In reflection of this, most of the legal disputes at the time were a result of clients hiring staff from professional services companies in violation of non-hiring clauses.[27]

Over the course of the 1960s, a number of changes took place that gradually and haphazardly undermined this pre-modern regime. For my purpose, one of the most important developments was the gradual emergence of a new type of software artifact: the *software product*. Software products were standardised off-the-shelf programs that were sold separately, some in the hundreds, a few in the thousands, typically for between \$5,000 and \$100,000.[28] Software products were discrete commercial objects that could be used without modification by a large number of contractors with little or no customization.[29] While Applied Data Research's Autoflow (1965) and Informatics Mark IV (1967) are often considered the earliest and most influential software products, by 1967 the number of proprietary programs on the market had increased to over a 100, with sales of about \$4 million.[30] From a small number of companies at the beginning of the 1960s, there were reported to be over 3,000 independent software and service companies by 1968.[31] As a result, by the end of the 1960s the term 'software-industry' had taken on its present day meaning 'signifying commercial organizations engaged in the production of programming artefacts'.[32]

There were a number of factors that prompted the emergence of the software products industry in the later part of the 1960s. One important factor was the proliferation and growing capabilities of computers. The rapid increase in the number

[24] Dave Sturtevant, ADAPSO Reunion Workshop, 'Industry Image', recorded 4 May 2002, CHM Ref No. X4425.2008, Computer History Museum, 19.
[25] Gene Bylinsky, 'Help Wanted: 50,000 Programmers' (March 1967) *Fortune* 141.
[26] Philip Stork, 'Legal Protection for Computer Programs: A Practicing Attorney's Approach' (1970) 20 *Copyright Law Symposium* 112, 115.
[27] ADAPSO Reunion Workshop, 'Contract Reference Directory', Computer History Museum (2002), CHM Ref No. X4410.2008, 14.
[28] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 3–4.
[29] Ibid.
[30] Anon, 'Software Gets a Hardsell Approach' (21 October 1967) Business Week 171. Donald H. Sundeen, 'General Purpose Software' (January 1968) *Datamation* 22.
[31] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 50.
[32] Ibid., 57.

of computers in the United States – estimated at 4,000 in 1960, 21,600 in 1965, and 48,500 in 1970[33] – created a number of software-related problems. One of which was that the software contractors who wrote bespoke programs for corporate and government customers were unable to keep up with the growing demand for custom-built software: simply put, computers were growing faster than programmers. This led to a concern about the shortage of software applications and programmers. These problems were compounded by the fact that the sharing organizations who supplied free software were criticized for being too hardware focused and because they did not share important and costly programs.[34] There was also a concern about the quality of the programs being created. Another problem was that many computer users lacked the in-house expertise to develop or customise software. And for those organizations that had the expertise to write software themselves, there were concerns about the cost of in-house production: it was often difficult to predict in advance how long it would take and how much it would cost to develop software. Pre-packaged software that was sold off the shelf at a fixed price helped to satisfy many of these concerns.

Another important factor that facilitated the emergence of the software products industry in the 1960s was the development of the common technical standards that interoperable pre-packaged objects require. One of the factors that had prevented the development of pre-packaged software in the early 1960s was the diversity of different standards then in use. For example, in 1960 IBM had at least seven different software incompatible computer architectures, each of which required unique operating systems and utilities. The situation changed in the mid-1960s, however, when IBM introduced the 360 family of computers. One of the features of the new 360 system, which consisted of fourteen different computers, many of which sold in large numbers, was that all of the computers used the same architecture. In so doing, IBM created a base-standard and a technical platform for the industry as a whole. As one independent software producer said, the establishment of 'a single architectural standard' facilitated by the 360 family of computers 'gave us a great customer base to sell to'.[35]

Yet another factor that played a role in the development of the software products market was the antitrust actions that were brought against IBM by the US Department of Justice and Applied Data Research. While there may be questions about the reasons for the unbundling, there is little doubt that it had a substantial impact on the nascent computing industry. One of the consequences of the unbundling that took place in the early 1970s was that software was marketed and sold separately from hardware. By helping to 'condition customers to pay for software' and by challenging the idea that software was a free good, the decision by IBM and

[33] Ibid., 50.

[34] Robert F. Brothers and Alan M. Grimaldi, 'Prater and Patent Reform Proposals' (1969) 17 *Catholic University Law Review* 389, 392; Editors Readout, (June 1966) *Datamation* 21.

[35] Lee Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 20.

other hardware companies to stop giving software away for free helped to create 'a vibrant market for software products, which previously had been merely embryonic. It was a turning point for the industry'.[36] As well as benefiting existing computer service and software firms such as Applied Data Research and Informatics, who saw a dramatic increase in sales after IBM announced that it would market and sell hardware and programs separately,[37] the unbundling of software also acted as a catalyst for new organisations to enter into the software products market. In this sense, unbundling was a 'crucial inflection point' in the development of the software products industry.[38]

The emergence of the software product industry not only saw a change in the way software was created, distributed, and used, it also changed the way people thought about software. For my purposes, the most important consequence of the development of the software product industry was that it changed what was expected or demanded of the law. While there had previously been little or no need for intellectual property protection, suddenly intellectual property was potentially relevant. Indeed, one of the consequences of the growth in the software product market was that it triggered a debate about the potential role that intellectual property might play in relation to software-related subject matter.

INTELLECTUAL PROPERTY PROTECTION FOR THE SOFTWARE PRODUCTS INDUSTRY

Early interest in the potential application of intellectual property to protect software was driven by two groups. The first were the financial institutions who loaned money to software companies. One of the concerns that banks and other financial institutions had when loaning money to software product companies was that many of the new start-up companies had very few assets other than the software that they were creating. The problem here was that the banks were uncomfortable loaning money purely on the basis of intangible assets. One of the strategies that the banks adopted to deal with this problem was to demand that software companies take out intellectual property protection over their software. This allowed the banks to point to the copyright or patent registration as if it was a tangible manifestation of the ephemeral software. The problems banks had with software's intangibility and the way that they dealt with this is captured in the comments of an industry representative about his experience in obtaining a loan from a bank at the time. As he said, the bank was 'alarmed that the principal software of the company had not been registered in the Copyright Office'. To remedy this, as a condition of the loan the bank

[36] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 89.

[37] Ibid., 115.

[38] J. Yates, 'Application Software for Insurance in the 1960s and Early 1970s' (1995) 24(1) *Business and Economic History* 123.

insisted that the 'company file all of those pieces of software in the Copyright Office so that they would have a lien on a registered copyright'.[39]

The second group agitating for protection were lawyers. For some lawyers, ensuring that software products were protected was an integral part of what it meant to be a lawyer. As Irving Kayton said when opening a software law conference at George Washington University in 1968: 'lawyers must protect their client's property rights or they will not have clients or, indeed, practice law'.[40] For other lawyers, legal protection was an integral part of what it meant for something to be a product. As one legal commentator noted, the 'one essential ingredient of the package concept [or software product] is a means of protecting the program: for without such protection, it ceases to be a commodity'.[41] For most lawyers, however, the primary reason why legal protection was needed was to prevent software from being pirated. Here, lawyers either worked on first principals – arguing that no one would invest in software unless it was protected[42] – or cited other lawyers about the manifest need for protection. Whatever the justification, the message was clear: without protection, software was vulnerable; 'the degree of competition and inevitably the quality of the end product, will be diminished'.[43]

One of the things that underpinned the various pleas for legal protection for software was a belief that software piracy was a problem that needed to be solved. In a sense it was presumed that the development of software products as discrete commercial objects necessarily created a need for intellectual property protection. While the separation of intellectual outputs from the people who generate them often creates a need for intellectual property protection, this is not necessarily the case (as lawyers at the time seemed to presume). Ultimately, the question of whether intellectual property protection is needed in a given situation depends on a range of factors from how easy it is to reproduce or copy the creative output in question and whether copying is seen as a problem, to whether other means are available

[39] Oscar H. Schachter, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 13.

[40] Irving Kayton, 'Foreword' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 8.

[41] David Bender, 'Trade Secret Protection of Software' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 3. One question that needs consideration is the role lawyers played in creating an expectation that software needed to be protected. Martin Goetz attended a session chaired by Mort Jacobs at a 1964 Spring Joint Computer Conference in Washington on 'Patents and other legal problems relating to Electronic Computers'. Martin Goetz, 'Memoirs of a Software Pioneer' (January–March 2002) *IEEE Annals of the History of Computing* 43, 50.

[42] As Whitlow Computer Systems, a company engaged exclusively in the development, production and sale of computer programs said, many potential investors had refused to invest in Whitlow 'simply because it could not give assurance that its computer programs will be held to be patentable subject matter. Brief Amicus Curiae for Whitlow Computer Systems, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 2 n 2.

[43] David Bender, 'Trade Secret Protection of Software' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 6.

to prevent unwanted copying or imitation. It was the later point that is relevant here. This is because while the emergence of software products did shift the focus of attention away from programmers towards the products of their labour, it did not (immediately) affect the relationships that existed between creators/producers and users/consumers of software. As had been the case with software contractors who had built strong relationships with their customers, software product firms also managed to establish close relationships with the users of their software. These relationships were reinforced by the pre-and after-sale support (including product customization, user training, and regular updates) that software products firms regularly provided to customers.[44] This was particularly the case with application software companies, who were more service companies than packaged goods companies and therefore worked closely with customers.[45] One of the consequences of this was that the use/misuse of software was largely controlled through personal ties and business-to-business relationships. As a software industry representative explained, the data processing managers who the software companies dealt with were 'not going to cheat because he could end up losing his job. So there was very little thievery'.[46]

The upshot of this is that despite the suggestions by lawyers at the time, there was little need for legal protection (at least to prevent piracy). Indeed, one of the things that software industry representatives looking back on the 1960s have stressed is that, in spite of what the lawyers may have suggested, piracy or as it was known at the time, thievery was not a problem. These sentiments were captured in the comment by the former president of the software products group at Dun & Bradstreet, Leo Keet, when he said:

> It was just a bizzarre time. The word I would use is paranoia. We, as an industry … were paranoid about things that didn't happen. We thought that there was going to be a lot of thievery. We wanted to anticipate it because we put so much of our money and intellectual energy and effort into building these things, and we didn't want them stolen.
>
> It wasn't until the PC industry came along [in the 1980s] that it actually turned into a huge problem. I can't emphasise that enough. I don't think you'll find anybody from the era of the 1960s and 1970s that will tell you that they had a big problem with thievery.[47]

[44] Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, MA: MIT Press, 2004), 6.

[45] Lee Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 14.

[46] Martin Goetz, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 16.

[47] Lee Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 14. 'The big concern we initially had was … unauthorised copying of software. But for us, it turned out, that rarely proved to be a problem … Piracy was not an issue for us. Remember this was not PC software where theft is a significant issue. This was all mainframe and mid-range stuff'. Dick Thatcher, ADAPSO Reunion Workshop, 'Contract Reference Directory' (2002) Computer History Museum CHM Ref No. X4410.2008, 14

While there may have been very little unauthorized use of programs by corporations[48] and even less by consumers,[49] this does not mean that intellectual property protection was not needed. One of the areas where this was the case was to protect software product firms from the predatory behaviour of the large hardware firms, particularly IBM. Rather than needing to prevent end-user piracy of software, software product firms needed legal protection to enable them to compete against hardware manufacturers and thus to get a share of the rapidly expanding market. The concern here was that without protection, software product firms were 'unable to compete with machine manufacturers who would be able to copy the programs with impunity and distribute them "free" with their machines'.[50]

One of the most vocal proponents of intellectual property protection for software was Martin Goetz, president and founder of Applied Data Research.[51] As we saw earlier, one of the earliest and most successful software products was Applied Data Research's Autoflow software program, which was designed to produce program flowcharts automatically. Applied Data Research, who had invested over US$4 million in software systems,[52] believed that one of the reasons for the low sales of its Autoflow software was because IBM had begun to offer for free a program called Flowcharter that also generated flowcharts automatically. As Goetz complained, the 'IBM Flowcharter became the major reason for a delayed or lost Autoflow sale. Our prospects went to IBM and asked for improvements to free IBM programs and it was widely believed IBM would develop a similar type of program and provide it to their customers for free'.[53] In response Goetz not only brought an antitrust action against IBM, he also joined with many others to argue that the only way that smaller software firms could protect themselves against the predatory behaviour of large hardware manufacturers was to ensure that software was given some type of legal protection.

Irrespective of whatever doubts there might be about whether legal protection was needed at the time, there is no doubt that there was a growing interest across the

[48] Martin Goetz, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 12. 'ADR was never aware of any company that was using [their] software without being authorized to use it'. Ibid., 16.

[49] 'I never had a customer steal from me'. Leo Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 11.

[50] Morton C. Jacobs, 'Commissions Report (re: Computer Programs)' (1967) *Journal of the Patent Office Society* 372, 376.

[51] As Goetz said, 'I got indoctrinated very early by Mort Jacobs, my patent attorney, who had previously worked at the Patent Office and then worked at RCA and then he was in private practice'. Martin Goetz, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 10.

[52] Martin A. Goetz, 'Protecting Computer Program Concepts and Copies' (1970) 14 *Idea* 7.

[53] Martin A. Goetz, 'How ADR Got Itself into the Software Products Business and Found Itself Competing against IBM', Computer History Museum (1998), 2. See also Martin Goetz, 'Memoirs of a Software Pioneer', (January–March 2002) *IEEE Annals of the History of Computing* 43, 50–53. Potential customers 'questioned why they should pay for an outside product when they could acquire software for "free" through IBM or an industry trade group'. Robert Head, 'The Travails of Software Resources' (January–March 2002) 82 *IEEE Annals of the History of Computing* 84.

1960s in the potential role that intellectual property might play in protecting software, an interest that was heightened by changes in labour law that made it increasingly difficult to control the movement of employees.[54] While the first copyright registration for software was granted in 1964,[55] there was little industry interest in using copyright to protect software-related innovations.[56] (There was even less interest in trade secret protection, which was thought to provide ineffective protection).[57] While the hardware manufacturer's interest in using copyright increased over time, independent software companies consistently showed little interest in using copyright to protect software. There were a number of reasons for this, including uncertainty about whether the registration of software would be upheld by the courts, the limited protection that was available if software was in fact protectable (given that it only protected the expression of programs), and uncertainty about how software should be represented for the purposes of registration. These problems were compounded by the fact that the Copyright Office (initially) did not accept object code for the purposes of registration. Instead, applicants had to submit source code and provide the full scope of the program (this was later changed so that applicants were only required to file 'pieces of the program'[58]).

Unsatisfied with the protection offered by copyright and trade secrecy, it was believed that patents offered the only viable mode of protection for software. In arguing for patent protection, a number of familiar arguments were rehearsed. In particular, it was argued that patent protection would stimulate investment in innovation, promote the continued creation and circulation of software, overcome the growing shortage of programmers, and help to counter the culture of secrecy that the business environment encouraged. In response to the argument that protection

[54] Employee's 'non-compete agreements were gradually being obviated by the courts, especially on the West Coast. California eventually made them useless.' Lee Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 16.

[55] The first copyright registration, was granted to John Banzhaff III under the 'rule of doubt' that favoured protection in 1964. John F. Banzhaf III 'Copyright Protection for Computer Programs' (1964) 14 *Copyright Law Symposium* 118; 'Copyright Registration for Computer Programs' (1963) 11 *Bulletin of the Copyright Society of the USA* 361. The Register of Copyrights accepted computer programs for registration provided that they contained sufficient original authorship, they had been published, and that the copies submitted for registration were in machine readable form.

[56] IBM was slow to support copyright because 'they were originally calling all their programs a service that they were giving away and putting in the public domain'. Leo Keet, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 12. However IBM eventually embraced copyright. From 1964 to January 1977, IBM and Burroughs were said to account for 971 of the 1,205 programs registered. *Final Report of the National Commission on New Technological Uses of Copyrighted Works: July 31, 1978* (Washington: Library of Congress, 1979), 34.

[57] See, e.g., David Bender, 'Trade Secret Protection of Software' (1969–70) 38(5) *George Washington Law Review* 909.

[58] Oscar H. Schachter, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 10.

was not needed and that software should continue to be given away for free, the proponents of patent protection cast doubts over the quality of the software that was shared at no cost. They also suggested that the free exchange programs only provided access to less important programs and that they did not include the valuable programs that might help competitors (such as multi-million-dollar airline reservation programs).[59]

Interestingly, the push for patent protection was also closely tied up with a desire to change how people thought about software which, in turn, was tied up with the professionalisation of the emerging industry. One of the problems at the time was that software was looked down on as a 'second-class citizen'; something that was attributed to the fact that software 'started out being a free service in the public domain'. Here, patenting was seen as a means 'for elevating the view of what software was. It shouldn't be free, it should be patentable'.[60] As Goetz explained, 'really … what we were trying to do' in seeking to patent software 'was to get stature' …. 'Every other industry seemed to have patent protection but here was an industry where you couldn't get patent protection.'[61] Patenting was also seen as a means of enhancing the reputation of the firms that produced software. It was suggested, for example, that Bell Laboratories' support for software patent protection was motivated by a desire for more public recognition in the programming area. The rational here was that if Bell's 'patents appears on programs that find wide use, [Bell] would become known as a source of programming excellence'.[62]

The push for patent protection for software was met with a hostile response from a range of parties. Somewhat surprisingly this included the Patent Office (who we would now expect to champion patent protection) and IBM (who for many years were reported to have 'the most patents of any company in the US, or in the world', but were against the patenting of software[63]). A number of arguments were made against patent protection for software. These ranged from general complaints that patent protection would stifle innovation and be counterproductive to the industry's growth and development to more specific concerns about the ability of the Patent Office to cope administratively with software patenting. In reflection of the close connection that existed between software and hardware, it was also suggested that

[59] Morton C. Jacobs, 'Commissions Report (re: Computer Programs)' (1967) *Journal of the Patent Office Society* 372, 376.

[60] ADAPSO History Program: Interview with Martin Goetz (3 May 2002) (interviewed by Jeffery R. Yost), 8.

[61] Martin Goetz, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 7. Patents were said to have a 'status' that copyright lacked. Calvin N. Mooers, 'Computer Software and Copyright' (March 1975) 7(1) *Computing Surveys* 45, 64. Martin Goetz, 'Memoirs of a Software Pioneer: Part 2' (October–December 2002) 14 *IEEE Annals of the History of Computing* 22.

[62] James P. Titus, 'Pros and Cons of Patenting Computer Programs' (February 1967) 10(2) *Communications of the ACM* 126.

[63] Martin Goetz, ADAPSO Reunion Workshop, 'Intellectual Property' (2002) Computer History Museum, CHM Ref No. X4589.2008 (Recorded 4 May 2002), 5.

if patents were granted over software, it would create problems for computer users who, given the intangible nature of software, would not know whether they were infringing someone else's patent.[64] In this sense, it was argued that patent protection on programs would restrict the ability of someone purchasing a computer from using the instructions built into it.[65] It was also argued, somewhat ironically, that patent protection was leading to more restrictive information handling practices by users and software companies. In particular, it was said that 'certain professional journals have now taken the position that they will no longer publish allegedly novel algorithms if the person who claims them claims patent protection'.[66] The opponents of patent protection also cautioned against changing something that they believed was already working well, indeed so well that it was 'difficult to conceive how the field could grow faster'.[67] Specifically it was said that as the rapid growth and innovation in software development that had taken place across the 1960s had occurred in an 'atmosphere of free and open exchange of computer program ideas' that protection was simply not needed.[68] In light of this it was suggested that the best strategy was to continue to give software away for free.[69]

For the most part, the arguments for and against the patenting of software in the 1960s and 1970s are familiar; they have been repeated in one form or another for a range of different types of subject matter over time. The situation is less familiar, however, when we shift to look at the contrasting ways software was perceived within the information technology industry and what this meant for the law.

THE 'CONTESTED ONTOLOGIES OF SOFTWARE'

While some of the older classes of patentable subject matter such as kaleidoscopes, steam engines, or dyes may now seem odd or quaint, it is relatively easy to compile a list of the different types of subject matter that have been presented to the law for evaluation over the years: recent examples include synthetic biology, AI-generated

[64] It was argued that if patents were patentable, each user of a computer would have to 'proceed at peril' in using a computer, since they never be able to know whether the algorithm used in the program was covered by an existing patent … leading to nuisance infringement actions. Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 14.

[65] Memorandum of IBM before the Patent Office on the Guidelines, 15; cited in Morton C. Jacobs, 'Commissions Report (re: Computer Programs)' (1967) *Journal of the Patent Office Society* 372, 378.

[66] Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 12.

[67] Letter from Donald Turner (Ass. Attorney General, Antitrust Division), to Edward J. Brenner (Commissioner of Patents) (21 October 1966) (cautioning against patent protection), as cited in Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 11.

[68] For discussion see Brief Amicus Curiae for the American Patent Law Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 20.

[69] Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 10–11.

inventions, nanotechnology, and genes. Although with hindsight it may be relatively easy to identify the subject matter that was under consideration at a particular point of time, when new forms of subject matter are first presented to the law for scrutiny, there is often confusion about what the subject matter should be called, what its defining features are, and how it compares to other types of subject matter. Given that would-be classes of potential subject matter are almost by definition novel, this is not surprising. What is more surprising, however, is that in some situations the law has found it difficult to determine what the subject matter in question is. This was and remains the case with software-related inventions.

One of the reasons why there were so many problems associated with the patenting of software is because as Nathan Ensmenger said, software is quintessentially a heterogeneous technology: meaning that software is 'inextricably linked to a larger social-technical system that includes machines (computers and their associated peripherals), people (users, designers and developers), and processes (the corporate payroll system, for example)'.[70] As would-be subject matter, software's heterogeneity presented problems for the law. The reason for this is that when determining the standing of a class of potential subject matter, patent law cannot and does not embrace an open-ended view of techno-scientific objects. Instead, when determining the standing of a class of subject matter, patent law needs to reduce the open-ended, fluid, and heterogeneous technology into something that is both closed, demarcated, and predictable and, at the same time, flexible enough to accommodate variations across the class of subject matter as well as changes that occur in the subject matter over time.

There were a number of reasons why patent law found software-related subject matter problematic. One reason for this was that software was defined negatively as those computer-related things that were not hardware. Indeed, in the 1959 article where the term was first used, John Turkey referred to software as those elements of a typical computer installation that were not 'tubes, transistors, wires, tapes and the like'.[71] The difficulty of defining something that was already defined in opposition to what it was not helped to contribute to software's 'widespread, ill-defined use'.[72] The difficulties that arose in ascertaining the contours of the subject matter were compounded by software's intangibility or immateriality[73] which meant, amongst other things, that there were no obvious traces or markers that could be relied upon to demarcate the boundaries of the subject matter.

While these factors were important, but often not in the way that we might first think, perhaps the most important reason why the law experienced so many

[70] Nathan Ensmenger, 'Software as History Embodied' (January–March 2009) 31(1) *IEE Annals of the History of Computing* 88.

[71] John Turkey, 'The Teaching of Concrete Mathematics' (1958) 65(1) *American Mathematical Monthly* 1, 9.

[72] Thomas Haigh, 'Software in the 1960s as Concept, Service, and Product' (January–March 2002) 24(1) *IEEE Annals of the History of Computing* 5.

[73] Unlike hardware which has visible boundaries to demarcate and define it.

problems in determining the ambit of software-related subject matter was because there was a fundamental disagreement within the nascent information technology industry about the way that the subject matter should be approached. This is important because as the history of patent law shows techno-scientific communities have not only consistently provided the law with potential new candidates for protection, they have also provided the means to allow the law to describe, demarcate, and identify that new subject matter. The presentation of new types of subject matter for legal scrutiny, whether organic chemicals, new plants, or mechanical innovations, has typically been accompanied by a shared understanding of what the subject matter is amongst the scientific and technical communities that generated it. What is so interesting about patent law's engagement with software-related subject matter is that this was not the case.

While there was an expectation (or hope) that the information technology community would help the law in dealing with the nascent subject matter, this did not occur. In part, this was because there were two contrasting ways of thinking about software-related subject matter that coexisted at the time: what Gerardo Con Diaz called the 'contested ontologies of software'.[74] While there was agreement that the fate of software turned on 'technological facts',[75] the parties were largely talking at cross purposes. This is because hardware manufacturers and software product companies did not agree on what the subject matter should be, let alone how it should be construed: they had very different understandings both about what the subject matter was and also about how it was to be interpreted. In particular, while hardware manufactures and their supporters argued that the debate should be about the patenting of computer programs, software companies argued that the debate should be about the patenting of programmed or special purpose computers as machines.

For hardware manufacturers, who were largely happy with the legal status quo, software-related subject matter was presented in such a way that it would not be patentable.[76] This was done by arguing that discussions about patentable subject matter should be limited to discussions about whether *computer programs* were patent eligible. In this context, programs were presented as 'nothing more than a set of instructions to a computer as to how it should manipulate information and data'.[77] Specifically, programs were presented as flat, inert, two dimensional descriptions of a process that 'specifies, in greater or lesser detail, the manner in which something

[74] Gerardo Con Diaz, 'Contested Ontologies of Software' (2016) 38(1) *IEEE Annals of the History of Computing* 23.

[75] Morton C. Jacobs, 'Patents for Software Inventions: The Supreme Court's Decision' (January 1973) 55 *Journal of the Patent Office Society* 59.

[76] Steven W. Usselman, 'Unbundling IBM: Antitrust and the Incentives to Innovation in American Computing' in (ed) Sally H. Clarke, Naomi R. Lamoreaux, and Steven W. Usselman, *The Challenge of Remaining Innovative: Insights from Twentieth-Century American Business* (Stanford, CA: Stanford University Press, 2009), 261.

[77] Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 6.

may be implemented.[78] In this sense, it was argued that computer programs were essentially immaterial creations that had a 'certain ephemeral … non-physical or non-machine character'.[79] Importantly, the descriptive character of computer programs remained 'the same whether the language used is binary, mnemonic assembly language or even higher level languages. It also remains the same regardless of the recording media, whether it be paper, punched cards, magnetic tape or even the internal magnetic cores of a computer memory. In all of these cases, the program continues to be explicative, that is, descriptive'.[80]

By limiting the subject matter to computer programs and by presenting computer programs as inert two-dimensional descriptions of processes, hardware manufacturers were able to argue that computer programs were non-patentable mental processes. Specifically, it allowed them to suggest that a program, like a punched paper piano roll, was 'nothing more than a set of instructions for the machine (i.e., computer or piano) automatically to implement the mental processes or steps contained in the algorithm or musical composition. Such creativity as exists lies solely in the development of the algorithm or musical composition and any patent issuing thereon would necessarily be grounded on the ideas or mental steps involved'.[81] By limiting the subject matter to static two-dimensional programs that merely specified the manner in which something could be implemented, hardware manufacturers were able to argue that a program was 'no more the subject matter of patent application than is the schematic diagram of an electrical circuit'.[82] While the subject matter here had a technical dimension, it primarily reflected the idea of the program as a commercial commodity. It was also an object protected by copyright but not by patents.[83]

While hardware companies argued that the question to be asked was whether computer programs were patentable subject matter, software companies such as Applied Data Research argued that discussions about patentable subject matter should focus on computer-related subject matter as machines. As Morton Jacob

[78] 'A Case History: Benson and Talbot: Appellant's Position: Computer Programs in General', Appendix C, appended to Robert O. Nimtz, 'Computer Application and Claim Drafting under Current Law' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 261. IBM argued that a 'computer program is simply a mode of expressing ideas'. Brief for Amicus Curiae International Business Machines, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 3.

[79] Morton C. Jacobs, 'Patentable Machines: Systems Embodiable in Hardware or Software (The Myth of the Non-Machine)' in (ed) Irving Kayton, *The Law of Software* (George Washington University, 1968) B-77, B-85, 1.

[80] Robert O. Nimtz, 'The Data Processing Revolution' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 128.

[81] Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 8.

[82] Ibid.

[83] *United States* v. *IBM* 69 Civ. 200 (SDNY 1969); *Applied Data Research* v. *International Business Machines Corporation* 69 Civ. 1682, (filed 22 April 1969).

said, a 'computing machine is clearly a "machine" within the statutory classes of 35 USC 101'.[84] 'Computer programs are parts of such machines, in fact, they are control mechanisms for the computer: as such, they are "machine" devices or an article of manufacture within the terms of 35 USC 101.' On this basis Jacob said: 'Once we recognize that the subject matter of these computer-program inventions is that of machines, we appreciate that the classical principles apply, and the issue of patentable subject matter under the Constitution or under the patent statutes is not really involved at all.'[85]

Unlike hardware companies who presented intangible computer programs and tangible hardware as discrete and separate objects, software producers argued that the subject matter only made sense when the program and the machine were combined.[86] While a programmable machine such as a general-purpose computer had potential, on their own these protean machines were 'merely a "warehouse" of unrelated parts'.[87] It was only when the program and hardware were combined to form a special purpose machine that the potential was able to be fulfilled.[88] That is, it was only when the computer was combined with the program that these 'moronic machines' were 'capable of accomplishing such varied jobs as corporate payrolls and Apollo moon shots.'[89] When loaded with a specific program that 'transfers the latent power of the theoretically general-purpose machine into a specific tool for solving real-world problems',[90] a computer becomes a special purpose machine; for example, 'an inventory control machine, a tax-return machine, a machine for automatically controlling a factory such as an oil refinery, a medical diagnosis machine, an engineering design machine for performing various calculations and for designing other machines etc etc'.[91]

[84] Morton C. Jacobs, 'Patentable Machines: Systems Embodiable in Hardware or Software (The Myth of the Non-Machine)' in Irving Kayton (ed), *The Law of Software* (George Washington University, 1968) B-77, B-85. 1.

[85] Ibid.

[86] Ibid. Morton C. Jacobs, 'Computer Technology (Hardware and Software): Some Legal Implications for Antitrust, Copyright and Patents' (1970) *Rutgers Journal of Computers and Law* 50, 52. While these arguments drew upon patent law's longstanding recognition of the patentability of combination claims to claim the combination which the union of the program and the computer creates, there was very little reference to this jurisprudence. One notable exception is Max W. J. Graham Jr, 'Process Patents for Computer Programs' (1968) *California Law Review* 466, 472–480 (arguing that the protection was ineffective).

[87] Edward J. Brenner, 'Guidelines to Examination of Programs' (9 August 1966) 829(2) *Official Gazette of the United States Patent Office* 442.

[88] General purpose computers 'can do anything for which we can provide suitable instruction … that is the source of its power'. However, 'precisely because it can do anything, it can do nothing in and of itself. It does things only when we provide the programs that cause the universal machine to emulate particular machines of our design'. Michael S. Mahoney, 'What Makes the History of Software Hard' (July–September 2008) *IEEE Annals of the History of Computing* 8, 10.

[89] William D. Smith, 'Fighter for Computer-Program Patents' (29 December 1968) The *New York Times* 19.

[90] Nathan Ensmenger, *The Computer Boys Take Over: Computers: Programmers, and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2010), 5.

[91] Morton C. Jacobs, 'Computer Technology (Hardware and Software): Some Legal Implications for Antitrust, Copyright and Patents' (1970) *Rutgers Journal of Computers and Law* 50, 51.

For software producers, what made the modern electronic digital computers unique and the reason why they differed from piano players and jacquard looms was their 'ability to be reconfigured via software into a seemingly infinite number of devices … it is the ability to be programmed via software that … encapsulates the essence of modern computing'.[92] While a piano roll would never change a player piano into anything but what it is, computers were universal machines that could 'be programmed to perform an almost infinite range of operations from a musical synthesizer and a payroll system through to an airline reservation system, as a classically designed machine'.[93]

When viewed functionally, the addition of a software program to control a general-purpose computer was said to 'be just as much a machine addition to it as the additional hardware programming'. In both cases, the addition of programming results in a machine that is different from the original. One reason for this was that a programmed computer was said to be '*structurally* different from the same machine without the program since its memory elements are differently arranged'.[94] In this sense, software producers argued that by 'programming a computer, the user creates a new machine'.[95] As Robert Nimtz explained, '[d]uring the actual execution of a program, a logical process is taking place or a new logical machine is taking form … During such execution, a new logical machine is formed and new logical processes are carried out on that new machine. Generally speaking, it is these new extant machines and extant processes that are the subject matter of patent claims'.[96] This way of viewing the subject matter enabled software producers to argue that the programmed computer acquired a new function recognizable by patent law. Importantly, this meant that the subject matter was potentially patentable.

While software companies argued that placing a different program into a computer fundamentally changed the nature of that computer, hardware companies consistently argued that a computer remained the same machine irrespective of the program that was used to operate it. As IBM said, the programming of a computer 'does not vary the actual nature of the computer so as to constitute a patentable invention'.[97] The idea that a computer remained the same whether or not it

[92] Nathan Ensmenger, *The Computer Boys Take Over: Computers: Programmers, and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2010), 5.

[93] Paul E. Ceruzzi, *Computing: A Concise History* (Cambridge, MA: MIT Press, 2012), 56.

[94] George A. Heitczman, 'Computer Programs Are patentable' (1970) 113(1) *Seton Hall Law Review* 113, 127.

[95] Brief for Amicus Curiae Institutional Networks Corporation, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct Term, 1971), 4. The programmed computer was 'structurally different from the same machine without the program since its memory elements are differently arranged'. George A. Heitczman, 'Computer Programs Are Patentable' (1970) 113(1) *Seton Hall Law Review* 113, 127. See George V. Elgroth, 'Software and Patent Law' (1966) *Patent Law Annual* 1.

[96] Robert O. Nimtz, 'The Data Processing Revolution' in *Software Protection by Trade Secret, Contract, Patent: Law, Practice, and Forms* (Washington: Patent Resources Group, 1969), 129.

[97] Brief Amicus Curiae on behalf of the Business Equipment Manufacturers Association, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 3.

was programmed was highlighted in the oral argument before the Supreme Court in *Gottschalk* v. *Benson* where in response to Justice White's question 'When the computer is programmed … it is not the same machine as it is when it isn't programmed?', the Government Attorney replied: 'It is precisely the same machine, Mr Justice White. It is precisely the same machine'.[98] As the Government Attorney noted, 'That is precisely the heart of our case'. The digital computer is 'really no more than an extension of an adding machine or calculator'. He added:

> Well, Mr Justice White, the analogy which we use in our brief – and I think that this is the appropriate analogy – is an old piano player which carries out – which plays songs when piano rolls are inserted into it. We do not believe that the computer acquires a new function every time it carries out new calculations that it is inherently built to perform, any more than a player piano carries out a new use every time a new piano roll is inserted into it.[99]

By arguing that a computer was the same machine irrespective of whether it contained a new and different program, hardware manufacturers were able to argue that the 'computer does not acquire a new function, in any sense recognizable by the patent law, every time it is programmed to perform a different set of arithmetical calculations, any more than a piano played acquires a new function each time it plays a new song'.[100] In this sense hardware manufacturers were able to argue that the programming of a computer was no more than a conventional and unpatentable use of a known machine, similar to placing a new piano roll in a player piano. In both cases, the end result was patent ineligible. As IBM said, the programming of a computer 'does not vary the actual nature of the computer so as to constitute a patentable invention'.[101] This, in turn, allowed the hardware manufacturers to assert that 'computer-program inventions relate to things other than machines and therefore are non-patentable'.[102]

Over the course of the 1960s and 1970s, hardware and software companies repeated their strategic and self-serving arguments about the nature of software-related subject matter in a range of venues including conferences, academic journals, trade magazine, policy reviews, newspapers, and amicus curia briefs (to both the Court

[98] Cited in Morton C. Jacobs, 'Patents for Software Inventions: The Supreme Court's Decision' (January 1973) 55 *Journal of the Patent Office Society* 59, 60 (transcript of oral arguments, 19).

[99] Ibid.

[100] Reply Brief for the Petitioners, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 5. As the Petitioners in *Gottschalk* v. *Benson* argued (including the US Solicitor General and the USPTO), a program in a computer was no different to a conventional use of a known machine, 'comparable to the insertion of a new piano roll in an old piano player'. Brief for Petitioners, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 17.

[101] Brief for Amicus Curiae International Business Machines, *Gottschalk* v. *Benson*, Supreme Court of the US, No. 71–485 (Oct. Term, 1971), 3.

[102] Morton C. Jacobs, 'Patentable Machines: Systems Embodiable in Hardware or Software (The Myth of the Non-Machine)' in (ed) Irving Kayton, *The Law of Software* (George Washington University, 1968) B-77, B-85, 1.

of Customs and Patent Appeals and the Supreme Court). In so doing they not only highlighted how important the task of deciding what the subject matter was, they also highlighted how entrenched and divided the industry's response was to this question. In a sense, the issue that underpinned these debates was whether or not the subject matter had been dematerialised. As we will see in Chapter 6, this had important ramifications for the way that patent law interacted with computer-related subject matter.