Intangible Inventions: A History of Software Patenting in the United States, 1945–1985

GERARDO CON DIAZ

On March 31, 2014, at the United States Supreme Court, Justice Stephen G. Breyer asked a lawyer named Carter G. Phillips to imagine King Tutankhamun, the ancient Egyptian pharaoh, sitting on top of a pyramid containing all his gold.[1] Breyer instructed Philips to suppose that King Tut, who had the habit of handing out vouchers for free gold, had hired a man with an abacus to keep tabs on his wealth. This ancient accountant was responsible for telling the king to stop handing out vouchers as soon as the total amount of gold given away surpassed whatever amount was initially stored in the pyramid. Essentially, his job was to say "Stop" as soon as the difference between the two amounts became zero or less. In this situation, Breyer explained, King Tut had simply used a human being to implement a very simple abstract idea: to say a word when a value reaches a limit.

doi:10.1017/eso.2017.36

Published online October 6, 2017

Gerardo Con Diaz is an assistant professor of Science and Technology Studies at the University of California, Davis, and an Affiliated Fellow of Yale Law School's Information Society Project. He holds a PhD in History (History of Science and Medicine) from Yale University, as well as an MPhil in History and Philosophy of Science from the University of Cambridge (Trinity College) and a BA in Mathematics from Harvard University. E-mail: condiaz@ucdavis.edu

I am indebted for advice and encouragement to my dissertation adviser, Daniel Kevles, and to my dissertation committee—Naomi Lamoreaux, Mario Biagioli, William Rankin, and Nathan Ensmenger. The research and writing for this dissertation and summary were made possible by the Science and Technology Studies Program at the University of California at Davis, the UC Davis ModLab, the Charles Babbage Institute, the Institute of Electrical and Electronics Engineers, the Lemelson Center for the Study of Invention and Innovation, the Business History Conference, the Society for the History of Technology, the Special Interest Group in Computers, Information, and Society, and the Yale Program in the History of Science and Medicine. I am also grateful to the IEEE for allowing me to republish portions of my work for the *IEEE Annals of the History of Computing* in *Enterprise & Society*.

1. Oral argument, *Alice Corporation v. CLS Bank International*. https://www.oyez.org/cases/2013/13-298 (hereafter, Oral argument, *Alice Corporation*).

Phillips immediately realized that thinking about King Tut was essential to his success. The lawyer stood before the justices in representation of the Alice Corporation, an Australian firm dedicated to the acquisition of software patents for financial transactions.[2] Alice had sued CLS Bank International, a New York-based bank, for the infringement of four patents aimed at so-called electronic escrow services. These are transaction methods wherein a computer manages payments between two parties in order to minimize the risk that one of the parties will fail to uphold its part of the deal. In its defense, CLS had argued that Alice's patents were invalid because the inventions that they described were not patent-eligible.[3] Mark A. Perry, representing the bank, argued that merely requiring a computer implementation did not render abstract ideas such as those at the core of the Alice patents eligible for patent protection.

All the major players in the computer industry were watching. Earlier that year, firms such as Google, Amazon, Facebook, and Netflix had written amicus briefs arguing that Alice's patents were improperly broad; that is, that software patents should be limited to a "specific way of implementing an abstract idea."[4] Microsoft, Adobe, and Hewlett-Packard each argued that software implementations of processes and algorithms are patent-eligible, but that Alice had failed to disclose an implementation of this kind.[5] Advocacy groups, including the Software Freedom Law Center, the Free Software Foundation, and the Open Source Initiative—all distinguished by their strong opposition to intellectual property protections for software—filed briefs arguing that software comprised nothing but algorithms written "in human-readable terms," and that without specialized machinery, they were altogether ineligible for a patent.[6]

These groups had submitted forty-two briefs in total, and Breyer had read all of them.[7] He had chosen the caricature of King Tut because

2. A patent grants an inventor the right to exclude others from making or selling his or her invention. This write-up uses the term *software patent* in reference to patents issued for computer programs and their implementations. It serves as shorthand for a patent that protects a computer program.

3. The term *patent-eligible* refers to inventions for which Section 101 of the United States Code's (U.S.C.) 35th title provides protection. This includes any "new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof." §101, 35 U.S.C.

4. Brief of Google Inc. et al., No. 13-298. https://www.americanbar.org/content/dam/aba/publications/supreme_court_preview/briefs-v3/13-298_resp_amcu_google-etal.authcheckdam.pdf

5. Brief of Microsoft Corporation et al., No. 13-298. https://www.americanbar.org/content/dam/aba/publications/supreme_court_preview/briefs-v3/13-298_affirm_microsoft-etal.authcheckdam.pdf

6. Brief of Software Freedom Law Center et al., No. 13-298. https://www.americanbar.org/content/dam/aba/publications/supreme_court_preview/briefs-v3/13-298_resp_amcu_sflc-etal.authcheckdam.pdf

7. Oral argument, *Alice Corporation*.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

it helped the justice to ponder the role that a computer plays in a financial transaction. If King Tut somehow managed to use a programmed electronic computer instead of a man with an abacus, would the resulting financial management system be patent-eligible? Breyer believed that it would not, and the other eight justices agreed. On June 14, 2014, the Supreme Court handed down a unanimous decision in favor of CLS. The decision, delivered by Justice Clarence Thomas, favored CLS by ruling that requiring a generic computer implementation did not render an abstract idea eligible for patent protection.[8]

Since then, industry and legal commentators have noted that the country has entered a new era in the history of software patenting—one in which, as one blogger for the American Bar Association put it, it is now "open season on these patents."[9] Hundreds of writers have taken on the task of speculating what the future of software patenting in the United States will look like in the aftermath of *Alice*. Software firms might now turn to copyright law and trade secrecy more eagerly than ever before, or they might entirely abandon traditional intellectual property (IP) protections such as patents and copyrights. Perhaps patent agents will need to develop entirely new ways of drafting patent applications, at least until courts and the Patent Office cease to interpret *Alice* as an indictment on what software patenting had become. Perhaps license agreements will become even longer. The industry will continue to thrive, but it is unclear what it will mean to own a program in this new legal environment.

Despite this plentiful attention, several key questions have so far remained unanswered: How and why did program makers start securing patents in the first place? How have patents, and IP protections more generally, shaped the American computing industry? What have terms such as "software" and "software patents" meant over the years? More generally, what does the interface between the computing industry and the law reveal about the history of software?

My doctoral dissertation, "Intangible Inventions," answers these questions by arguing that the commercial, legal, administrative, and conceptual problems borne out of the patent protection of computer programs has shaped the emergence of software as a commodity, an invention, and a creative work. This argument stands at the intersection of the histories of technology, business, and law, and it is grounded on my study of corporate and federal archival materials, trade literature, oral histories, and an assortment of rare books and manuscripts.

8. *Alice v. CLS*. http://www.supremecourt.gov/opinions/13pdf/13-298_7lh8.pdf
9. Seidenberg, "Business-Method."

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

This summary sketches four key arguments that stand at the dissertation's core. In the form of three chronologically overlapping vignettes and one concluding reflection, these arguments illustrate how business history enables scholars to connect legal history with the history of computing. A substantially revised and enriched version of this work will be available (after spring 2019) in a book that I am currently writing for Yale University Press.[10]

Argument 1: The early patent protections available to computer programs hinged on the notion that code is equivalent to hardware

In 1946, after leaving the Manhattan Project, a mathematician named Richard Hamming began working on Bell Telephone Laboratory's projects on telephone switching.[11] He specialized in the use of computers to transform numerical data from one number system to another on a relay machine, the Mark V, developed at Bell. This machine had the habit of aborting its programs as soon as it encountered a processing error, so Hamming devised an error correcting code that would enable the Mark V to correct these errors on the go and avoid a complete stop.[12]

Hamming believed that his code was nothing more than mathematical algorithms, which rendered it ineligible for patent protection because of the so-called mental steps doctrine. This doctrine deemed that "mental steps," such as computing, comparing, and observing, are not subject matter eligible for a patent. However, Bell's lawyers disagreed. In 1948, they asked an electrical engineer named Bernard Holbrook to make diagrams that disclose the electrical circuits that Hamming's program produced in the computer.

The lawyers then filed an application for a patent called "Error Detecting and Correcting System."[13] This patent was aimed not at Hamming's codes, per se, but at a machine constructed according to the circuitry diagrams that Holbrook had produced. This is the first instance of the successful implementation of a patent-drafting technique

10. I have also published three articles based on my doctoral dissertation. See Con Diaz, "Text in the Machine"; Con Diaz, "Contested Ontologies of Software"; Con Diaz, "Embodied Software."

11. Portions of the text in this first section are drawn from Con Diaz, "Embodied Software." They are reprinted with permission from *IEEE Annals of the History of Computing*.

12. The observations on the Mark V's operation are grounded on Thompson, *Error-Correcting Codes*.

13. Richard Hamming and Bernard Holbrook, "Error-Detecting and Correcting System," US Patent 2,552,629, filed January 11, 1950, and issued May 15, 1951.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

that lawyers in the 1960s would call "embodying software"—that is, securing patent protection for a computer program by patenting a machine that worked in accordance with the program instead of attempting to patent the program itself.

This technique was grounded on the fact that in the late 1940s, the act of programming a computer could be a very physical act. For instance, the work of the women who programmed the ENIAC (the most notable electronic computer) consisted, literally, of rewiring the computer's circuits so that the machine could execute whichever program interested them.[14] More important, the aim of Bell Labs was not to develop and lease computer programs but to perform the research and development necessary to maintain and strengthen the Bell System.[15] This means that the impulse to secure patents for programs emerged not from the desire to profit from the programs themselves (as scholars used to assume) but instead from the patenting practices of the telecommunications industry.

Starting in the 1950s, Bell's patent-drafting technique spread to organizations ranging from oil and aerospace firms to hardware manufacturers. For the next two decades, it allowed their lawyers and patent agents to protect programs that controlled oil refineries, scientific instruments, automatic weapons systems, and, of course, data processing equipment.

Argument 2: The possibility of securing patent protections for software enabled the emergence and early growth of the software products industry

For many years, historians of computing have identified Martin Goetz's flowcharting application, AUTOFLOW, as the first software *product*, that is, the first computer program that a firm actually sold to its clients.[16] These historians highlight the commodification of AUTOFLOW as Applied Data Research's (ADR) quick response to a business opportunity in the mid-1960s. No one was selling software, so selling a flowcharting program—one of the most widely needed applications at the time—was an appealing idea.[17]

In contrast, "Intangible Inventions" shows that firms such as ADR embraced the idea of selling their programs not just because there was a demand for them but also because their managers and lawyers

14. Haigh, Priestley, and Rope, *ENIAC in Action*; Light, "When Computers Were Woman."

15. Millman, *History of Engineering and Science*.

16. See, for instance, Campbell-Kelly, *Airline Reservations*.

17. Excerpts from this section are drawn from Con Diaz, "Embodied Software."

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

started to believe in what I call the "gospel of software patenting." By this I mean the promise that patent law would give them the legal protections necessary to start selling physical copies of their programs and making inroads against IBM's market dominance without the fear of piracy.

In the mid-1960s, IBM announced its System/360 line of mainframe computers.[18] This was IBM's most popular machine until the 1980s. It was a so-called general-purpose computer—one that could be programmed to meet the needs of a broad range of users. At the time, IBM and other hardware makers distributed their programs by bundling them; that is, by providing them free of charge with hardware purchases or leases. The bundles for the System/360 included a wide variety of programs that performed commercial, scientific, accounting, and engineering functions.

The popularity of the System/360, paired with a general sense of dissatisfaction with IBM's software for it, created a demand for low-cost alternatives to IBM's programs. However, software companies at the time did not normally sell their programs. Instead, they developed them through special contracts or leased whichever ones appeared to be in high demand. Aiming to profit from the System/360's popularity, managers at these firms started to wonder if they could actually sell physical copies. However, they were hesitant to do so because they believed that a sales-based business model would require them to forego the anti-piracy clauses that they normally included in their leasing or custom-development contracts.

As firms such as IBM and RCA gravitated toward the idea of selling programs, so did many patent lawyers who used to work for hardware manufacturers and industrial research laboratories. These lawyers had eclectic backgrounds that included scientific and technical training, and they were very familiar with Bell's technique of disclosing programs as hardware. Among these lawyers was Morton Jacobs, who once worked for RCA, and who in the mid-1960s started attending software conferences and trade associations. There, he delivered talks to software firm managers to explain that patent law would enable them to abandon leasing contracts and sell their programs without fear of piracy.

One of the people who met Jacobs at one of these venues was Marty Goetz, from ADR. They became good friends, and they secured the patents that scholars have identified as the first software patents. What is more important is that Jacobs and Goetz filed a patent for

18. For the standard narrative on the System/360, see Campbell Kelly et al., *History of the Information Machine*.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

AUTOFLOW and started selling copies of it as widely as possible (see Figure 1). They were not stellar, but they did prove to ADR and its competitors that if they tried to sell their software, plenty of people would buy it. Hardware and software firms soon started filing patents for the programs they wished to sell. By the 1970s, the Patent Office and the courts were handling more applications aimed at computer programs than ever before.

Argument 3: The political economy of the computing industry has yielded mutually incompatible conceptions of hte nature of software as an invention

One of the main organizing principles of "Intangible Inventions" is what I term ontologies of software—by which I mean conceptions of the nature of software as an invention.[19] Legal scholars have shown that many judges who consider software patents have grounded their rulings on whichever ontology of software they tend to favor, and that this grounding can determine the outcome of a trial. In contrast, "Intangible Inventions" shows that ontologies of software are far more than just the byproduct of the idiosyncrasies or legal inclinations of

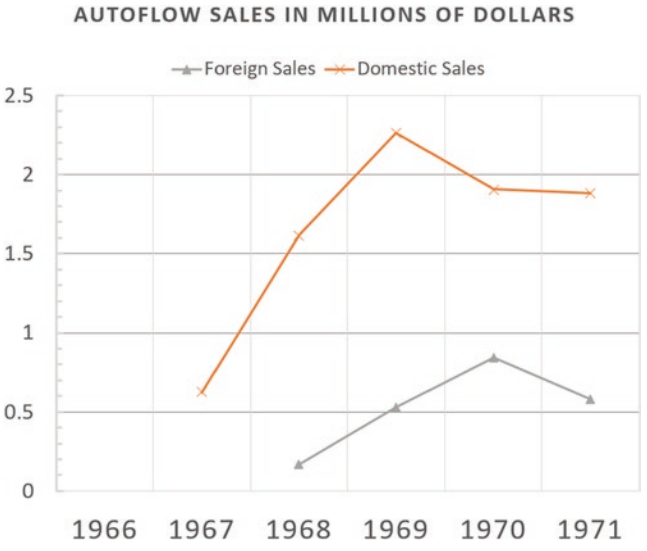

Figure 1 Autoflow sales (in $ millions).

Source: Created by the author, based on data at the Charles Babbage Institute.

19. See Con Diaz, "Contested Ontologies of Software"; Con Diaz, "Text in the Machine."

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

specific judges. They are, in fact, central components of the long-term legal and business strategies of computing firms. Each ontology of a software program is tied to the aims and needs of the firms that advances it.

Consider, for instance, the relationships between IBM and small software firms.[20] In 1969 IBM implemented its so-called unbundling: it started selling its programs instead of distributing them free of charge with the purchase or lease of hardware. Almost immediately, it launched a series of efforts aimed at driving smaller firms out of the market for software products. IBM's managers agreed that software patenting ran against their company's best interests. If software firms intensified their patenting efforts, then IBM might need to spend millions of dollars in licensing fees in order to develop programs that may or may not sell well. Because software firms were highly litigious, there was also a possibility that IBM would need to pay large sums in settlements or have the release of one of their programs delayed by a court.

To prevent either situation, IBM's lawyers set out to eliminate the patent protection of computer programs in all its forms. For the next decade, they argued in courts, in front of Congress, and at trade association meetings that software is nothing but text—something akin to a book or pamphlet and therefore the proper subject of copyrights, not patents. However, lawyers at software firms knew that copyrights offered weak protection because they were limited to the specific sequence of words and symbols that a programmer would write. For this reason, whenever IBM delivered a brief or testimony on the textual nature of software, the software firms' lawyers would follow by arguing that software is, in fact, a machine—a tangible object more similar to a car than to a book and therefore patent eligible.

More generally, Figure 2 maps out the ontologies of software that "Intangible Inventions" identifies as central to the history of software patenting. The purely textual or purely mechanical natures described above are the bottom two vertices. The vertex on the top corresponds to the notion of programs as mathematical algorithms, which mathematicians such as Hamming often held. This algorithmic conception became especially popular in the 1980s because the personal computing revolution allowed manufacturing firms of all sizes to computerize their industrial processing techniques. These firms took no issue with the claim that software was an algorithm because they aimed to secure patent protections for programs only indirectly, as one more element in a system of manufacture.

20. See Campbell-Kelly et al., *Computer*; Usselman, "Unbundling IBM"; Campbell-Kelly, *Airline Reservations*.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

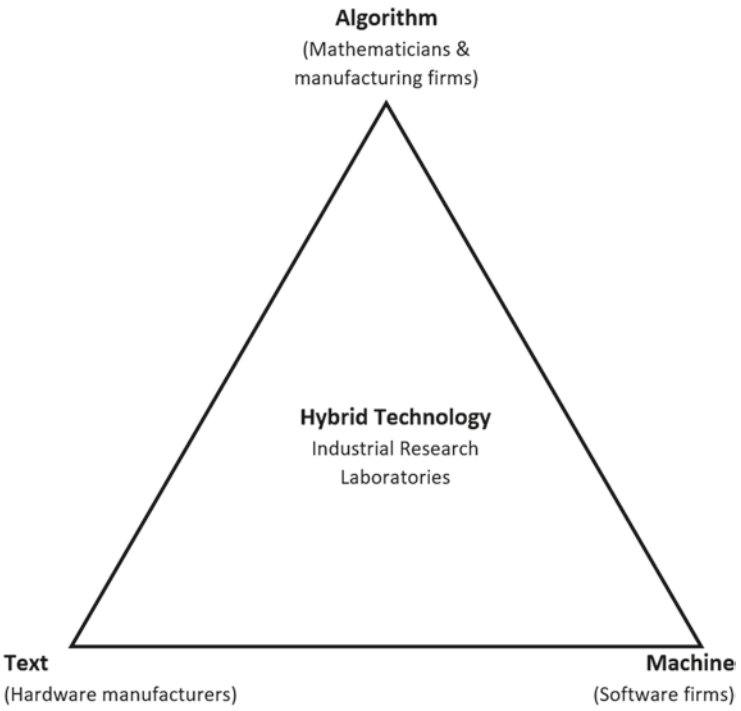

Figure 2 Schematic representation of the main ontologies of software central to the patent protection of computer programs until the 1980s.

Source: Created by the author, based on his doctoral dissertation.

Industrial research laboratories generally moved somewhere in the middle region. Their aim was not to sell any programs but to reduce their licensing costs without precluding the possibility of patenting their own software. Their lawyers often decided which ontology to advance based on the case at hand, but most of the time they argued that programs were hybrid technologies—part text and part machine or a text that became a mechanical implementation of an algorithm.

There is one important exception to this diagram. It relates to the computer hobbyists of the late 1970s: the people who tinkered with computers at places such as the Homebrew Computer Club and the People's Computer Company. Some of these hobbyists would become keen users of software patents and copyrights, but many of them would soon gravitate toward the ideals of free and/or open source software known today. Leading members of this second group considered each program to be a part of a series of layered routines and interfaces, not unlike an onion. At the core of the onion is the computer's circuitry; on its outside is the interface that enables a user to operate it, and in the middle are the many programs that mediate between a user's experience and the machine itself.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

This conception was central to the hobbyists' views on intellectual property. Indeed, they considered the sale and intellectual property protection of programs to be a great danger because corporate control of even one layer of the onion could preclude the free and unabated interaction between humans and machines.

Argument 4: The issues surrounding software patenting today are the outcome of a slow evolution of patentable subject matter that has been taking place for almost seventy years

From the 1940s to the 1990s, the Patent Office and courts at all levels handed down hundreds of interrelated decisions that bear on the patent protection of computer programs. Legal scholars and commentators have mined the web of legal reasoning that these decisions created in search of a culprit or a hero—a court decision, or perhaps a cluster of decisions, that enable software makers today to obtain very broad protection for their programs. More recently, news outlets often report how software patents run against the patent system's intended purpose of promoting innovation and encourage some firms to become litigation engines.

In contrast, "Intangible Inventions" shows that there is no single historical moment—let alone a recent one—to point to in order to support or oppose software patenting. This is neither a story of how a particular court or company suddenly ruined the system nor how courts systematically sided with big businesses to the detriment of users or small firms. Instead, it is the story of how people in a rapidly changing technological environment negotiated what software is and what it means to own and commodify it. In the process, they caused a slow expansion of what is now considered to be patent-eligible inventions and of the language that software makers can use to describe their work.

In its book form, my work will include not just the arguments above but also analyses of the international politics that influenced software patenting in the United States, the advocacy work against patents and certain forms of copyrights by proponents of free and open source software, the relationships between internet technologies and American patent law in the late twentieth century, and the impact of very recent cases such as *Alice v. CLS* on the contemporary software industry. Despite its broader geographic, chronological, and legal scope, this book will share with the dissertation one major lesson: the history of program making has carried with it a history of patent drafting, and their joint development has slowly transformed our understanding of software as something to be made, owned, and sold.

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press

Bibliography of Works Cited

Books

Campbell-Kelly, Martin. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry.* Cambridge, MA: MIT Press, 2003.

Campbell-Kelly, Martin, William Aspray, Nathan Ensmenger, and Jeffrey Yost. *Computer: A History of the Information Machine*. Colorado: Westview Press, 2013.

Haigh, Thomas, Mark Priestley, and Crispin Rope. *ENIAC in Action: Making and Remaking the Modern Computer*. Cambridge, MA: MIT Press, 2016.

Millman, S. (ed.). *A History of Engineering and Science in the Bell System: Communications Sciences (1925–1980).* AT&T Bell Laboratories, 1984.

Thompson, Thomas. *From Error-Correcting Codes through Sphere Packings to Simple Groups*. New York: Carus, 1984.

Articles, Book Chapters, Blogs

Con Diaz, Gerardo. "Contested Ontologies of Software: The Story of *Gottschalk v. Benson*, 1963–1972." *IEEE Annals of the History of Computing* 38, no. 1 (January–March 2016): 23–33.

———. "Embodied Software: Patents and the History of Software Development, 1946–1970." *IEEE Annals of the History of Computing* 37, no. 3 (July–September 2015): 8–19.

———. "The Text in the Machine: American Copyright Law and the Many Natures of Computer Programs, 1974–1978." *Technology & Culture* 57, no. 4 (October 2016): 753–779.

Light, Jennifer. "When Computers Were Woman." *Technology and Culture* 40, no. 3 (July 1999): 455–483.

Seidenberg, Steven. "Business-Method and Software May Go through the Looking Glass after Alice Decision." *ABA Journal* (blog), February 1, 2015. http://www.abajournal.com/magazine/article/business_method_and_software_patents_may_go_through_the_looking_glass_after

Archives

Charles Babbage Institute, University of Minnesota, Minneapolis, MN.

Court Cases

Alice Corporation Pty. Ltd. v. CLS Bank International et al., 573 U.S.__134 S. Ct. 2347 (2014).

https://doi.org/10.1017/eso.2017.36 Published online by Cambridge University Press