A Brief History of Software Patents (And Why They're Valid)

Adam Mossoff

SEPTEMBER 2013

CENTER FOR THE PROTECTION OF INTELLECTUAL PROPERTY

A Brief History of Software Patents (and Why They're Valid)

ADAM MOSSOFF

Today, there is significant public debate over patents on the digital processes and machines that comprise computer software programs. These are often referred to as "software patents,"[1] but this is an odd moniker. Aside from the similarly mislabeled debate over "DNA patents,"[2] nowhere else in the patent system do we refer to patents on machines or processes[3] in a specific technological field in this way; for instance, people do not talk about "automobile brake patents" or "sex toy patents" as their own category of patents deserving of approval or scorn. (Yes, there are sex toy patents, and there are infringement lawsuits in which none other than Judge Richard Posner, a strident critic of today's patent system,[4] ruled that a particular sex toy was obvious and therefore unpatentable.[5])

Unfortunately, the policy debates today about "software patents" are rife with extensive confusion and misinformation about what these patents are and even about what "software" is. Even the Court of Appeals for the Federal Circuit is deeply confused about these patents, as evidenced by its highly fractured en banc decision in *CLS Bank v. Alice Corp*.[6] In 135 pages of numerous concurring and dissenting opinions that accompany the one-paragraph per curiam majority opinion, the *CLS Bank* court threw patent doctrine in this booming, innovative industry into even more disarray.[7] Judge Lourie's concurring opinion, joined by a substantial number of his colleagues, essentially argues that computer programs are unpatentable.[8] In her dissenting-in-part opinion, Judge Kimberly Moore rightly observed that Judge Laurie's opinion (and the fractured *CLS Bank* decision itself) represents "the death of hundreds of thousands of patents, including all business method, financial system, and software patents as well as many computer implemented and telecommunications patents."[9] Commentators have been equally critical of *CLS Bank*.[10]

Given the widespread confusing rhetoric and the concomitant doctrinal upheaval, a little historical perspective can be helpful and illuminating. First, knowing the historical evolution of software patents—even in classic "potted history" form[11]—is important because it reveals that the complaints today about intellectual property (IP) protection for computer programs are nothing new. Opposition to IP protection for computer programs has long existed—predating the Federal Circuit's 1998 ruling that business methods are patentable,[12] predating the Federal Circuit's 1994 ruling that computer programs are patentable as the equivalent of a digital "machine,"[13] and predating the Supreme Court's 1980 decision that a computer program running a rubber vulcanization process was patentable.[14] In fact, computer programmers and others initially opposed extending copyright protection to computer software programs, as I will discuss shortly.

Second, this history reveals that the shift in legal protection from copyright law in the 1980s to patent law in the 1990s was not a result of strategic behavior or rent-seeking by commercial firms who exploited their access to the halls of power in Congress (or somehow duped the courts into providing them the same legal protections). To the contrary, this historical evolution from copyright to patent law represented a natural legal progression as the technology evolved from the 1960s up to the mid-1990s. As it happens in our common law system — precisely because it is designed to happen this way — legal doctrines evolve in response to changes in innovative technological products and commercial mechanisms that, through the marketplace, spread these new technological values throughout the world.

It bears emphasizing that this is a "potted history" (in a non-pejorative sense). In a short essay I cannot recount every historical detail, and space constraints will require me to compress some developments into a simplified version. Of course, one should consult more detailed historical accounts of the digital revolution and its follow-on revolutions. For example, I recommend T.R. Reid's *The Chip* (2001), which provides an engaging and accessible recounting of the scientific and technological developments that made the Digital Revolution possible.

What is a "Software Patent"?

Before we can address the history, though, it is necessary to get clear on what exactly we mean by a "software patent." One of the primary problems with the term "software patent" is that, like other widely used terms in the patent policy debates today,[15] it lacks an objective definition. For instance, many critics of "software patents" attack them as patents on "mathematics"[16] or patents on a "mathematical algorithm,"[17] but this is sophistry. As commentators have repeatedly recognized, a word processing program like Word for Windows or a spreadsheet program like Excel are not the same thing as 2+2=4,[18] and the fact that computer programs use mathematics is an argument that proves too much. All patented innovation uses mathematics; in fact, physicists love to say that the universal language of the universe is mathematics.[19] So if taken seriously, the argument that a "web browser, spreadsheet, or video game *is* just math and therefore it's not … eligible for patent protection,"[20] would invalidate all patents if applied equally to other inventions, especially processes and methods. All inventions of practically applied processes and machines are reducible to mathematical abstractions and algorithms (e.g., a patentable method for operating a combustion engine is really just an application of the law of PV=nRT, the principles of thermodynamics, and other laws of nature comprising the principles of engineering).

Complicating things even further, the term "software patent," even when it is not being used in a way that invalidates all patents, is often used to refer to many different types of patented innovation. The term has been used to encompass such inventions as electrical patents and business method patents simply because the patented innovation uses some type of computer software program in its implementation. (As discussed in Hal Wegner's famous patent law listserv shortly after the GAO Report was released, one concern with the GAO Report is its surprising, and what many think is unrealistic, claim that

As commentators have repeatedly recognized, a word processing program like Word for Windows or a spreadsheet program like Excel are not the same thing as 2+2=4, and the fact that computer programs use mathematics is an argument that proves too much.

[F]ew people realize the vast numbers of valid and valuable patents on computer programs. The entire Internet rests on patented innovation in computer programs: the packet-switching technology used to transmit information over the Internet was patented by Donald Watts Davies (Patent No. 4,799,258).

"By 2011, patents related to software made up more than half of all issued patents."[21] This only makes sense if one includes not just classic computer programs among total issued patents, but any and all inventions that require some type of computer program in their implementation.[22])

For ease of reference given the ubiquity of this term in the policy debates, I will refer to "software patents" in this essay, but I will limit this term solely to patents on a set of machine-readable instructions that direct a central processing unit (CPU) to perform specific operations in a computer.[23] In short, "software" means a *computer program*, such as a word processing program (e.g., Word), a spreadsheet (e.g., Excel), or even programs run on computers on the Internet, such as Google's search algorithm, Facebook, eBay, etc. Of course, the reality is far more complicated than this, but that's not the point of this essay.

In fact, few people realize the vast numbers of valid and valuable patents on computer programs. The entire Internet rests on patented innovation in computer programs: the packet-switching technology used to transmit information over the Internet was patented by Donald Watts Davies (Patent No. 4,799,258). Robert Kahn and Vinton Cerf, the inventors of the TCP/IP packet-switching protocol, later patented their follow-on invention of a packet-switching version of a knowbot[24] (Patent No. 6,574,628). Larry Page and Sergy Brin patented their famous search algorithm when they were graduate students at Stanford, and such patented innovation was a reason why Page and Brin received venture-capital funding for their start-up company, Google (there are several patents, but Patent No. 6,285,999 is one of the core ones). There are slews of other valid patents on technologically and commercially valuable computer programs, such as an early one from

1993 for one of Excel's core spreadsheet functions (Patent No. 5,272,628).

To understand why these and many, many other patents on computer programs are both valuable and valid, it is necessary to understand whence computer programs came, how they changed in both their technological and commercial function after the 1970s, and why patent law was extended to secure this technological innovation in the early 1990s.

The Digital Revolution

Our story begins in the early years of the Digital Revolution with the invention of the integrated circuit in 1958-1959 (independently invented by Jack Kilby and Robert Noyce).[25] At that time, "software," at least as we now understand this term, did not mean what we think this term means today. Software was designed for specific computers and only for those computers. To wit, what worked on a specific IBM mainframe did not work on a DEC minicomputer (which was the size of a refrigerator).

(A young reader might ask, "Who is DEC?" Good question, young man or woman! The Digital Equipment Corporation (DEC) was one of the early leading firms manufacturing computers in the high-tech industry in the 1960s, ultimately bringing in multi-billion dollar revenues.[26] Its founder and CEO, Ken Olson, was admired by a young Bill Gates.[27] Olson also infamously said in 1977, "There is no reason for any individual to have a computer in his home."[28] That's why DEC is no longer around and why young people today no longer remember this company.)

The Copyright Controversy

Despite the start of the Digital Revolution a mere 60 years ago, its early growing pains have become the equivalent of "ancient history." For this reason, many people no longer remember that the protection of computer programs under copyright—something accepted today as an allegedly "obvious" legal alternative to patent protection—was originally disputed rigorously by programmers and others. The question of whether computer programs were copyrightable was a tremendous flashpoint of controversy for much of the 1960s and 1970s, which is ironic given that people today blithely assert that we don't need patent protection for computer programs because "copyright protection … makes patent protection mostly superfluous."[29] (This claim is also false, as the historical development makes clear and as will be explained shortly.)

Despite substantial controversy, in 1964 the Registrar of Copyrights started to register copyright protection for software code for computer programs.[30] Although there was no direct legal challenge to the Copyright Registrar's decision to begin registering copyrights for computer programs, the public policy debates did not go away.[31] The controversy continued, especially in the courts, for almost two decades,[32] and it was not resolved until Congress enacted the Computer Software Copyright Act of 1980,[33] which specifically authorized the protection of software code by the Registrar of Copyrights under the Copyright Act. In sum, opposition to IP protection for computer programs has existed from time immemorial, regardless of whether it was copyright or patent.

The PC Revolution

It is significant that the Computer Software Copyright Act was enacted in the early 1980s because it was during this time—the late 1970s and early 1980s—that the PC Revolution began ("PC," for the uninitiated, means Personal Computer). This is the point in time that marks the shift away from hardware and software as a unified, single product, to hardware and software as distinct products. This is the revolution brought to us by the young hackers and computer geeks of the 1970s—Steve Jobs, Steve Wozniak, Bill Gates, Nathan Myrhvold, etc.—who conceived, designed, and implemented the idea of an operating system (OS) running on a CPU that could serve as the operational platform for any computer program written by anyone performing any tasks, such as playing tic tac toe or blinking lights on a circuit board in a certain pattern (just some of the original programs end-users could write and operate in the 1970s) to the sophisticated word processing, spreadsheet, and computer-assisted design (CAD) programs that began to be sold and used on PCs in the 1980s.

The significance of the PC Revolution is that computer software programs now became* separate products *that consumers could purchase, install, and use on their PCs.

Any programmer can easily replicate the GUI or other features of a commercially successful computer program—copying the valuable function of the program—without copying the literal software code that created this valuable function.

The significance of the PC Revolution is that computer software programs now became *separate products* that consumers could purchase, install, and use on their PCs (either an "IBM Compatible" or a Mac). In fact, computer programs came in a *box* that consumers physically took off shelves and purchased at checkout registers at retail stores, such as at an Egghead Software outlet. (Egghead Software closed all its retail stores in 1998 due to the dominance of the Internet as a medium over which to order DVDs, and, eventually, through which end-users now directly purchase and download in 30 seconds their new software products or apps.[34])

The significance of a computer program becoming a separate product is that the *value* in software, what the consumer was seeking in purchasing it from the retailer, was the *function* of the program as experienced by the consumer (called an "end-user" in high-tech parlance). For instance, it was the value in the ease of use of a graphical user interface (GUI) of a particular word processing program, such as Word for Windows, that made it more appealing to consumers than the text-based commands of older word processing programs, such as WordPerfect. Or it was the pull-down menu in a Lotus1-2-3, the first widely successful spreadsheet program. The end-user now had a word processing program with many functions in it, such as editing text, italicizing text, "cutting" and "pasting," changing margins for block quotes, etc. *This* was the value in the product sold to the consumers, and thus *this function* is what designers of computer programs competed over for customers in the marketplace. For instance, few people today remember the battle in the late 1980s and early 1990s between WordPerfect (a text-based word processor developed for the text-based command system of DOS) and Word (a pull-down menu and button-based "point and click" GUI word processor for the Windows and Apple GUI OS).

This is not a radical or novel insight; it is a mundane fact recognized by many who have worked in the high-tech industry for the past several decades. Back in 2006, Nathan Myhrvold recounted how even many people working in the high-tech industry did not think that a company that solely made software like Microsoft could succeed. In 1987, he explained that he attended a

> *big industry conference in the PC industry. And there was a panel discussion I participated in—"Can Microsoft Make it Without Hardware?" I swear. Now, we had a proposition and the proposition was that not only can you make software valuable without hardware; software was actually a better business without hardware, because if you separated yourself off and you just became a software company you could focus on making the software best….An independent software company can target everybody's stuff.*[35]

What Myhrvold means by "target[ing] everybody's stuff" is that a company like Microsoft could succeed in selling computer programs that provided functional value to a vast array of end-users. Thus, for instance, Robert Sachs, a patent attorney who specializes in high-tech innovation and serves as an evaluator for high-tech standards, explains that the "vast majority of value in software comes not from some deeply embedded algorithm that can be protected by trade secret. Rather, it comes from the creation of new functionality that has immediate and apparent value to the end user, whether that's a consumer or an enterprise."[36]

In the late 1980s and early 1990s, this amazing development in new technology and new commercial intermediaries in delivering new computer programs to consumers created a problem: any programmer can easily replicate the GUI or other features of a commercially successful computer program—copying the valuable function of the program—without copying the literal software code that created this valuable function. In sum, the code becomes distinct from the end-user interface or the function of the program itself.

And there's the rub (to paraphrase the Bard): copyright protects someone only against copying of their literal words, not the broader idea or function represented by those words. In copyright law, this is the well-known legal rule referred to as the idea/expression dichotomy (express words are protected under copyright, but ideas are not).[37]

It is also reflected in the equally hoary legal rule that copyright does not protect utilitarian designs.[38]

This issue was brought to a head in the famous copyright case of *Lotus v. Borland*.[39] Lotus, the creator and owner of the very famous spreadsheet program Lotus 1-2-3, sued Borland in 1990 for copying Lotus's innovative pull-down menus in Borland's spreadsheet program, Quattro Pro. Lotus's design of the pull-down menus in Lotus 1-2-3 —these are now standard in all GUI-based computer programs—made it very efficient to use and this was a major reason for its commercial success.

The *Lotus* case was active for five years, and ultimately resulted in a trip to the U.S. Supreme Court, which split 4-4 in affirming the lower court (Justice Stevens recused himself), and thus the Supreme Court didn't hand down a precedential opinion.[40] As a result of the 4-4 split, the lower appellate court's decision (the Federal Court of Appeals for the First Circuit) was affirmed by default. The First Circuit held that Lotus could not copyright its pull-down menus because these were a functional "method of operation," i.e., a utilitarian design, and not an expressive text capable of receiving copyright protection.[41] The First Circuit and the four Justices who affirmed the First Circuit were correct in applying long-standing and fundamental copyright doctrine in denying copyright protection to the *functionality* of a computer program.

By the mid-1990s, as represented in the famous *Lotus v. Borland* case, it was clear that copyright could no longer adequately secure the value that was created and sold in software programs by the fast-growing high-tech industry. The *value* in a software program is the *functionality* of the program, such as Lotus 1-2-3, Excel, WordPerfect or Word for Windows. This function was the reason why consumers purchased a program, installed it and used it on their computers, whether an Apple computer or a Windows machine. But this functionality could be replicated using myriad varieties of code that did not copy the original code, and copyright did not protect the functional components of the program that this code created for the end-user—and for which the end-user purchased the program in the first place.

The Shift to Patent Law

This simple legal and commercial fact—copyright could not secure the real value represented in an innovative computer program—explains why in the mid-1990s there was a shift to the legal regime that could provide the proper legal protection for the innovative value in a computer program: patent law. As the Supreme Court has repeatedly recognized in contrasting patents against other IP regimes, such as copyright and trademark, "it is the province of patent law" to secure "new product designs or functions."[42]

The value in a software program is the functionality of the program, such as Lotus 1-2-3, Excel, WordPerfect or Word for Windows. This function was the reason why consumers purchased a program, installed it and used it on their computers.

In fact, this shift from copyright to patent law in the mid-1990s mirrors the equally important shift in the early 1980s when the courts and Congress definitively extended copyright protection to computer programs at the start of the PC Revolution. At the time, neither legal development was destined to occur by necessity, but, in retrospect, neither development was a historical accident from the perspective of the continuing success of the Digital Revolution. These two legal developments served as the fulcrums by which it was possible for inventors and innovating firms, such as Apple, Microsoft, eBay, Google, etc. to commercialize these newly created values. (See, e.g., the earlier-cited patented innovation in computer programs, properly secured to these companies, which made it possible for them to bring such values to the marketplace and to everyone's lives.)

At approximately the same time that the First Circuit and Supreme Court came to the legally correct conclusion in *Lotus v. Borland* that the functional value in the pull-down menus was not copyrightable, the Court of Appeals for the Federal Circuit expressly recognized that computer programs were patentable as a digital "machine." In its now-famous 1994 decision in *In re Alappat*,[43] the Federal Circuit ruled that a specific computer program that performed a specific and identifiable function for an end-user was not an "abstract" claim to an unpatentable idea or "algorithm."[44] To the contrary, such computer programs were patentable inventions.[45]

In essence, the Federal Circuit recognized the basic truth to which many firms in the high-tech industry owed

their existence: a computer program such as the Excel spreadsheet program "is not a disembodied mathematical concept which may be characterized as an 'abstract idea.'"[46] A computer program, such as Google's search algorithm, or a sub-program, such as an operation in Excel's spreadsheet, is the digital equivalent of "a specific machine."[47] In sum, the invention of a word processing program is the equivalent in the Digital and PC Revolutions of the invention of a mechanical typewriter in the Industrial Revolution. Similarly, an e-mail produced by the functions of a word processing program in an email program, such as Outlook or Eudora, is the digital equivalent of a physical letter written by a typewriter and mailed via the U.S. Post Office to its recipient.

Again, similar to the identification that the value in a computer program is its functionality to the end-user, the identification of the essential functional similarity between a mechanical typewriter and a word processing program is not particularly insightful or radical. As any computer programmer will tell you, the functions of a program can be performed perfectly in either software or hardware; the functional operation between the two is a distinction without a difference, except that a computer program is less costly and more efficiently sold and used by end-users. In fact, this equivalence between hardware and software is exactly what happened for the first several decades of the Digital Revolution before the invention of the integrated circuit and before the PC Revolution. And for those of us old enough to remember the very first word processors, there was not much to them beyond what an electrical typewriter could do in the 1970s and 1980s (including correct spelling errors after a word was typed and other formatting functions as well).

In sum, the functionality of binary code in a specific computer program is in principle no different from the functionality achieved in the binary logic hardwired into computer hardware. The fact that both are easily identified by firms, retailers and end-users confirms that the two can be specific, real-world and useful products. This functional equivalence between hardware and software further reflects the fact that the difference between computer programs (either in software or hardware) and the mechanical machines they replaced is itself a distinction without a difference — both have been innovative inventions deserving of protection under the patent laws.

Conclusion

The Industrial Revolution gave us patented innovation in sewing machines,[48] typewriters, and telephones, and the Digital and PC Revolutions have given us patented innovation in word processors, email and ebooks. To restrict the patent system to only the valuable inventions of the nineteenth century is to turn the patent system on its head—denying today's innovators the protections of the legal system whose purpose is to promote and secure property rights in innovation.

In the words of the Supreme Court's recent decision in *Bilski v. Kappos*,[49] patent law is a "dynamic provision designed to encompass new and unforeseen inventions."[50] As the *Bilski* Court recognized, a physical-based "machine-or-transformation test may well provide a sufficient basis for evaluating processes similar to those in the Industrial Revolution—for example, inventions grounded in a physical or other tangible form. But there are reasons to doubt whether the test should be the sole criterion for determining the patentability of inventions in the Information Age."[51]

The American patent system has succeeded because it has secured property rights in the new innovation that has come about with each new era—and it has secured the same property rights for all types of new inventions, whether in the Industrial Revolution or in the Digital Revolution. It is time to leave behind sophistical rhetoric, such as "software patent," and recognize that computer programs are valuable inventions performing very real and valuable functions for consumers the world over. This is why people from all walks of life pay money to companies like Apple, Microsoft, Dell, Cisco and many others to purchase these programs. As made clear in *Borland v. Lotus*, this is a real-world value that cannot and should not be secured by copyright. It also cannot be secured by trade secret because the functions of a program are the publicly known capabilities sought by end-users (and over which high-tech companies compete for customers). As the history of the evolution of patent protection for computer programs makes clear, this valuable innovation can be secured only by the IP regime specifically designed to secure functional value in new technological innovation —the patent system.

ENDNOTES

1 *See* Wikipedia, *Software patent debate* (as of Sep. 19, 2013), http://en.wikipedia.org/wiki/Software_patent_debate.

2 *See* Adam Mossoff, *A Century-Old Form of Patent*, N.Y. Times, Jun. 6, 2013, http://www.nytimes.com/roomfordebate/2013/06/06/can-the-human-blueprint-have-owners/a-century-old-form-of-patent.

3 *See* 35 U.S.C. § 101 (providing that "any new and useful process, machine, manufacture, or composition of matter" is patentable).

4 *See* Richard A. Posner, *Why There are Too Many Patents in America*, The Atlantic (July 12, 2013).

5 *See Ritchie v. Vast Resources, Inc. (d/b/a Topco Sales)*, 563 F.3d 1334 (Fed. Cir. 2009) (Posner, J.).

6 717 F. 3d 1269 (2013) (en banc).

7 *See* Semil Shah, Op-Ed., *The Scale, Competitiveness, And Industrial Strategies in Mobile Computing*, TechCrunch, Sep. 8, 2013, http://techcrunch.com/2013/09/08/the-scale-competitiveness-and-industrial-strategies-in-mobile-computing/ (describing vibrant, dynamic growth in the computer industry, especially in the last two years).

8 *CLS Bank*, 717 F.3d at 1276-92.

9 *Id*. at 1301.

10 *See* John Kong, *The Alice in Wonderland En Banc Decision by the Federal Circuit in CLS Bank v. Alice Corp*, IPWatchdog (May 14, 2013, 3:16 pm), http://www.ipwatchdog.com/2013/05/14/the-alice-in-wonderland-en-banc-decision-by-the-federal-circuit-in-cls-bank-v-alice-corp/id=40344/.

11 *See* MacMillan Dictionary, http://www.macmillandictionary.com/dictionary/british/potted (as of Sep. 16, 2013, 10:21 PM GMT).

12 *State St. Bank & Trust Co. v. Signature Financial Grp*., 149 F. 3d 1368 (Fed. Cir. 1998).

13 *In re Alappat*, 33 F. 3d 1526 (Fed. Cir. 1994) (en banc).

14 *Diamond v. Diehr*, 450 U.S. 175 (1981).

15 *See* Adam Mossoff, *The SHIELD Act: When Bad Economic Studies Make Bad Laws*, Center for the Protection of Intellectual Property Blog (Mar. 15, 2013), http://cpip.gmu.edu/2013/03/15/the-shield-act-when-bad-economic-studies-make-bad-laws/ (identifying how "patent troll" lacks any definition and is used non-objectively in patent policy debates).

16 *See* End Soft Patents, *Software is math* (as of Sep. 19, 2013), http://en.swpat.org/wiki/Software_is_math.

17 This characterization of computer programs as merely "mathematical algorithms" is an unfortunate byproduct of the Supreme Court's decision in *Gottschalk v. Benson*, 409 U.S. 63 (1972), in which Justice William O. Douglas described an invention of a fundamental software program for running all computers as an "algorithm." *Id.* at 65 ("A procedure for solving a given type of mathematical problem is known as an 'algorithm.' The procedures set forth in the present [patent] claims are of that kind."). Justice Douglas thus concluded that the invented computer program was an unpatentable abstract idea:

> It is conceded that one may not patent an idea. … The mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is

affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself.

Id. at 71-72. This was an unfortunate misinterpretation of the nature of computer programs as such, and it has caused much confusion in patent law about both computer programs and what makes them patentable inventions. *CLS Bank* simply represents the nadir of this confusion. What is notable, as is made clear in this essay, is that this confusion about the nature of computer programs in 1972 was perhaps understandable, if only because the PC Revolution had not yet occurred and thus it was much harder for judges to understand what made computer programs valuable as separate (patentable) inventions from the computer hardware on which they ran.

18 *See* Gene Quinn, *Groklaw Response: Computer Software is Not Math*, IPWatchdog (Dec. 15, 2008, 6:30 am), http://www.ipwatchdog.com/2008/12/15/computer-software-is-not-math/.

19 Carolyn Y. Johnson, *A talk with Mario Livio – Is Mathematics the Language of the Universe*, Boston Globe, Feb. 8, 2009, http://www.boston.com/bostonglobe/ideas/articles/2009/02/08/a_talk_with_mario_livio/.

20 Timothy B. Lee, *Software is Just Math*, Forbes, Aug. 11, 2011, http:/www.forbes.com/sites/timothylee/2011/08/11/software-is-just-math-really/.

21 U.S. Gov't Accountability Office, GAO-13-465, INTELLECTUAL PROPERTY: Assessing Factors That Affect Patent Infringement Litigation Could Help Improve Patent Quality 21 (2013), http://www.gao.gov/products/GAO-13-465.

22 *See Patent Statistics in the GAO Report*, High Tech Intellectual Property Legal Blog (Sep. 18, 2013), http://blog.hiplegal.com/2013/09/gao_softwarepatents/("The problem, of course, is that there are no 'exclusively software' classes. So if an entire patent class is counted, it is extremely likely to include non-software cases as well."). Wegner and others also claim that the GAO actually made an outright error in its counting methodology. *See* Hal Wegner, *GAO Patent Litigation Report (con'd): "[P]atents related to software ma[ke] up more than half of all … patents*, LAIPLA (Aug. 2, 2013), http://www.laipla.net/gao-patent-litigation-report-cond-patents-related-to-software-make-up-more-than-half-of-all-patents/ ("The GAO authors apparently counted 20,000 software patents instead of 2,000 under the methodology at p.12 n.27 (explaining Figure 1). Thanks to Greg Aharonian for sharing this information with the patent community."); http://www.global-patent-quality.com/GRAPHS/SoftElec.htm (reporting Greg Aharonian's statistics on issued patents that show that even the 2,000 number is almost twice the actual rate of issuance of "software" patents).

23 *See* Wikipedia, *Software* (as of Sep. 19, 2013), http://en.wikipedia.org/wiki/Software.

24 *See* John Markoff, *Creating a Giant Computer Highway,* N.Y. Times (1990), http://www.nytimes.com/1990/09/02/business/creating-a-giant-computer-highway.html.

25 *See* T.R. Reid, *The Chip: How Two Americans Invented the Microchip and Launched a Revolutio*n 76-80 & 91-95 (2001).

26 *See* Wikipedia, *Digital Equipment Corporation* (as of Sep. 19, 2013), http://en.wikipedia.org/wiki/Digital_Equipment_Corporation.

27 Bill Gates has written: "An inventor, scientist, and entrepreneur, Ken Olsen is one of the true pioneers of the computing industry. He was also a major influence in my life and his influence is still important at Microsoft through all the engineers who trained at Digital and have come here to make great software products." Chloe Albanesius, *Computing Pioneer Ken Olson Dead at 84*, PC Magazine, Feb. 8, 2011, http://www.pcmag.com/article2/0,2817,2379648,00.asp.

28 *See* Joelle Tessler, *Kenneth Olsen, Pioneering Founder of Computer Company, Dies at 84*, Wash. Post, Feb. 9, 2011, http://www.washingtonpost.com/wp-dyn/content/article/2011/02/09/AR2011020906305.html.

29 Timothy B. Lee, *The Supreme Court Should Invalidate Software Patents*, Forbes, Jul. 28, 2011, http://www.forbes.com/sites/timothylee/2011/07/28/the-supreme-court-should-invalidate-software-patents/.

30 *See* National Commission on New Technological Uses of Copyrighted Works, Final Report 82 (1979); Copyright Office Circular 31D (Jan. 1965).

31 *See, e.g.,* Allen W. Puckett, *The Limits of Copyright and Patent Protection for Computer Programs*, 16 Copyright L. Symp. 81, 104-05 (1968) (recognizing that there is limited copyright protection for some aspects of computer programs but that "[s]ource programs embodied in punch cards or magnetic tape present a doubtful case"); Pauline Wittenberg, Note, *Computer Software: Beyond the Limits of Existing Proprietary Protection Policy*, 40 Brooklyn L. Rev. 116, 117-18 (1973) ("With respect to computer software, such questions [about patent or copyright protection] have been under discussion in both legal and trade journals and in the courts for nearly a decade; no clear answers have emerged.").

32 *Compare Data Cash Systems, Inc. v. JS&A Group, Inc.*, 480 F. Supp. 1063 (N.D. Ill. 1979) (holding object code is not copyrightable) *and Tandy Corp. v. Personal Micro Computers, Inc.*, 524 F.Supp. 171 (N.D. Cal. 1981) (holding object code in ROM is copyrightable). *See also Synercom Tech., Inc. v. Univ. Comp. Co.*, 462 F. Supp. 1003, 1014 (N.D. Tex. 1978) (holding software code "formats" is not copyrightable).

33 Pub. L. No. 96-517, 94 Stat. 3015, 3028 (1980).

34 *See* Wikipedia, *Egghead Software* (as of Sep. 19, 2013), http://en.wikipedia.org/wiki/Egghead_Software.

35 Nathan Mryhvold, *Invention: The Next Software*, Intellectual Ventures, at 5 (2006), http://www.intellectualventures.com/assets_docs/Invention_Next_Software_Transcript_2006_Speech.pdf.

36 Robert R. Sachs, *Applying Can Openers to Real World Problems: The Failure of Economic Analysis Applied to Software Patents*, Bilski Blog (Aug. 13, 2013), http://www.bilskiblog.com/blog/2013/08/applying-can-openers-to-real-world-problems-the-failure-of-economic-analysis-applied-to-software-pat.html.

37 *See Baker v. Seldon*, 101 U.S. 99, 104 (1879) ("[T]he teachings of science and the rules and methods of useful art have their final end in application and use; and this application and use are what the public derive from the publication of a book which teaches them. But as embodied and taught in a literary composition or book, their essence consists only in their statement. This alone is what is secured by the copyright."); *Morrisey v. Proctor & Gamble Co.*, 379 F.2d 675, 678 (1st Cir. 1967) ("Copyright attaches to form of expression ….").

38 *See Baker*, 101 U.S. at 102 ("[N]o one would contend that the copyright of the treatise would give the exclusive right to the art or manufacture described therein. … That is the province of letters-patent, not copyright. The claim to an invention or discovery of an art or manufacture … can only be secured by a patent from the government.").

39 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided Court*, 516 U.S. 233 (1996).

40 *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 516 U.S. 233 (1996).

41 *Lotus*, 49 F.3d at 815.

42 *Qualitex Co. v. Jacobsen Products Co., Inc.*, 514 U.S. 159, 164 (1995); *Elmer v. ICC Fabricating*, 67 F.3d 1571, 1580 (Fed. Cir. 1995) ("patent law, not trade dress law, is the principal means for providing exclusive rights in useful product features"). *See also Baker,* 101 U.S. at 102 ("[T]he exclusive right to the art or manufacture …. is the province of letters-patent, not copyright.").

43 33 F. 3d 1526 (Fed. Cir. 1994) (en banc).

44 *Id.* at 1545.

45 *Id.*

46 *Id.* at 1544.

47 *Id.*

48 *See* Adam Mossoff, *The Rise and Fall of the First American Patent Thicket: The Sewing Machine War of the 1850s*, 53 Ariz. L. Rev. 165 (2011).

49 130 S. Ct. 3218 (2010).

50 *Id.* at 3227.

51 *Id.*

ABOUT THE AUTHOR

Adam Mossoff is Professor of Law and a Senior Scholar and Co-Director of Academic Programs at the Center for the Protection of Intellectual Property at George Mason University School of Law. He has published extensively on the history of patents, with his articles appearing in the *University of Pennsylvania Law Review*, *Cornell Law Review*, *Boston University Law Review*, and other journals. His article on the first patent war, the Sewing Machine War of the 1850s, has become an important part of the public policy debates concerning patent litigation, patent licensing, and patent pools, having been cited in the GAO Patent Litigation Report (August 2013). He has presented his research at many academic conferences, congressional briefings, and professional conferences, as well as at the PTO, the FTC, the DOJ, and the Smithsonian Institution. He thanks Matt Barblan, Ron Katznelson, Michael Risch, and Robert Sachs for comments and edits on earlier drafts of this essay. Steven Tjoe and Wen Xie also provided research and copy-editing assistance.

CENTER FOR THE PROTECTION OF INTELLECTUAL PROPERTY

The Center for the Protection of Intellectual Property (CPIP) at George Mason University School of Law is dedicated to the scholarly analysis of intellectual property rights and the technological, commercial and creative innovation they facilitate. Through a wide array of academic and public policy programming, including conferences, roundtables, fellowships, debates, teleforum panels, podcasts, and other events, CPIP brings together academics, policy makers and stakeholders in the innovative and creative industries to explore foundational questions and current controversies concerning patents, copyrights and other intellectual property rights. Ultimately, CPIP seeks to promote balanced academic discussions grounded in rigorous scholarship and to inform the public policy debates on how securing property rights in innovation and creativity is essential to a successful and flourishing economy.

CENTER FOR THE
Protection of Intellectual Property

George Mason University School of Law
3301 Fairfax Drive
Arlington, VA 22201

http://cpip.gmu.edu

Check out our blog at
http://cpip.gmu.edu/blog


Visit us on Facebook at
www.facebook.com/cpipgmu


Follow us on Twitter @cpipgmu

GEORGE MASON UNIVERSITY | School of Law