CHAPTER ONE

Introduction

How should software be protected from undue imitation and plagiarism? At present, all of the traditional means of protecting intellectual property (IP)—patents, copyright, and trade secrets—are applied to software in one manner or another, and the U.S. Congress has even invented a new type for cases in which these may be insufficient, via the Digital Millennium Copyright Act.

Software is not just like any other machine, as some courts have ruled, and it is not just *Hamlet* with numbers: it is a functional hybrid that can be duplicated at no cost, is legible by computers in some forms and by humans in others, and has a unique mathematical structure. All of these facts have to be taken into consideration in designing any type of IP protection for software.

Patents

It has become a hobby among computer scientists to find the worst software patents granted. There are hundreds that make a competent programmer groan—and want to file for his or her own patents. For now, a single example will suffice to illustrate why software authors and users are so bothered by the state of patents today.

Patent 6,389,458, granted May 14, 2002 (filed October 30, 1998, by Brian Shuster), is for pop-up browser windows, which are typically used

Figure 1-1. This Opens a New Window and Puts It in the Foreground

```
function onExit(){
   popup = window.open("pop.html","Don't go!");
   popup.focus();}
```

by advertisers to put ads on top of the content that people actually want to see, and to make it difficult for users to leave a web site. Figure 1-1 shows the three lines of code required to implement the patent in JavaScript, a language included in web browsers since December 1995, although the patent also covers implementations in any other programming language, even ones that have not yet been invented.[1]

The U.S. Patent and Trademark Office (USPTO) deemed that this combination of one line of code to open a computer window and a subsequent line to focus on the window is a new, nonobvious invention, and that no persons may put these three lines of code in sequence in their own work unless they pay Shuster's company (Ideaflood, Inc.) a royalty for the privilege of doing so. In 2018 this combination of three lines will enter the public domain.[2]

A Counterpoint

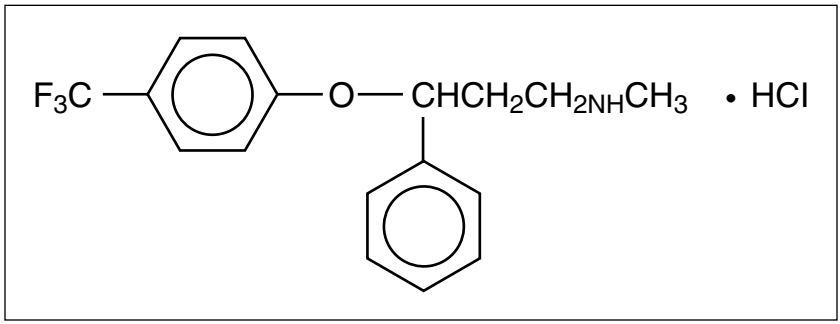
Here is the abstract for patent 4,314,081 (granted February 2, 1982, to Bryan Molloy and Klaus Schmiegel): "3-Aryloxy-3-phenylpropylamines and acid additions salts thereof, useful as psychotropic agents, particularly as anti-depressants." This patent covers the active ingredient in the formula for the antidepressant Prozac, shown in figure 1-2.

It is unlikely that even the best chemists could look at the chemical formula in figure 1-2 and infer that it could alleviate symptoms of depression in certain people. If they could, it would be because they had studied the work of Molloy and Schmiegel. Nor could we ask the best chemists to

1. For those who would like to infringe this patent on their own web pages: include the JavaScript from figure 1-1 and the tag `<BODY onUnload="onExit()">` in the HTML for the page.

2. A patent is valid for twenty years following the date when the application was filed. The patent can be viewed at the USPTO's website, www.uspto.gov. Given a patent number, the patent can also be downloaded from pat2pdf.org.

Figure 1-2. This Alleviates Depression



Used with permission of Eli Lilly and Company.

quickly jot down a chemical compound to alleviate depression and expect them to produce anything like this formula, the product of years of research by Molloy and Schmiegel costing untold dollars. Conversely, Shuster's invention would make a good quiz question for an undergraduate computer science class. Nonetheless, the patent for Prozac and the patent for pop-ups are entirely equal under the law.

A Persistent Problem

The pop-up patent is not an isolated case that slipped by an overworked patent examiner. Systematic differences in how software and machines or chemicals are constructed cause software patents to be systematically overbroad or obvious.

To paraphrase Socrates, the unexamined patent is not worth giving. Yet most software patent advocates claim as a platitudinous truth that software is just like any other technology. For example, in his 57-page review of IP protection for software, Kenneth Dam, Brookings scholar and IBM's former vice president of law and external relations, devotes one sentence to software versus physical patents: "In principle, the economic issues involved in software-related patents raise no economic issues other than those presented by patents generally."[3] He then discusses more general problems about the patenting system without a word of evidence to demonstrate that the economics of software is just like the economics of all other technologies.

3. Dam (1995).

If his claim were true, software patents would certainly make sense. In reality, the economics of software differs significantly from the economics of all other fields. Although some of software's problems have analogues in other industries (see chapter 5), many are almost entirely unique, notably the problems stemming from its mathematical properties, the structure of the software market, and the importance of interoperability (see chapters 3, 6, and 7, respectively).

Outside of a foolish consistency, there is no reason for patent law to ignore these unique features. If there were separate laws for physics- and chemistry-related inventions, the courts would be tied up for decades attempting to determine which laws applied where. But software is so clearly different from physical machines (I draw the line precisely and unambiguously in chapter 4) that the courts and USPTO could readily maintain an appropriately drawn line.

In assuming that there is no such difference and thus extending patent protection to software, courts have overlooked three important distinctions. First, a sufficiently detailed description of a computer program is the program itself, so it is sometimes difficult to distinguish between the idea and its implementation. For the pop-up window, the idea is a window that automatically opens and moves to the front when the user views a new page; the implementation is listed in figure 1-1. For Prozac, the idea is a selective serotonin reuptake inhibitor (SSRI); the implementation is shown in figure 1-2. Traditionally, patents have been granted to implementations of ideas and not to the ideas themselves—there are a dozen SSRIs on the market that did not infringe on the Prozac patent. But in software, the pattern has been reversed: most patents cover ideas like the pop-up window, regardless of implementation, so they tend to be too broad.

Second, a program is, in a literal sense, a piece of mathematics. This is not merely a play on words or a loose metaphor; a basic theorem of computer science demonstrates their equivalence. The courts agree that pure math is not patentable but that software is—yet the two are equivalent. The courts dumped the problem of reconciling the contradiction on the USPTO, which has resolved it by allowing patents on mathematical algorithms.

Third, vastly different categories of people write software. Nobody makes drugs but drug companies, so a patent on Prozac is a restriction only on other drug companies. But a patent on a piece of code is a restriction not only on software companies but also on the information tech-

nology department of every company in America, not to mention every person who writes macros to facilitate his or her work, or even students who (unlike chemistry students) could easily write a patent-infringing program and distribute it online.[4] Because software patents are a restriction not only on competitors but on a wide array of computer users, the cost-benefit analysis underlying patent law needs to be done anew for software.

The Problem Has Come to the Fore

Although the argument thus far may seem abstract, the economic consequences of bad patents are very real. To date, the USPTO has granted between 170,000 and 200,000 software patents, and applications continue to flood in; each one of those issued gives the holder the right to sue others where no such right existed before.[5] Because independent invention is not a defense against claims of patent infringement, anyone working in front of a computer could be a target for a profitable infringement suit. Some entrepreneurs have responded to this bonanza of lucrative targets by creating businesses, such as Acacia Technologies, whose sole purpose is to buy software patents and sue companies for infringement.[6] Because the nature of the software writing process makes independent invention much more common than in other fields, opportunistic lawsuits have been more numerous as well.

In an interview with venture capitalists and software developers, Ronald Mann, co-director of the Center for Law, Business, and Economics at the University of Texas at Austin, repeatedly found a resigned attitude toward patents:

> Software patents are multiplying so rapidly that it is likely that many product startups that are developing ultimately will infringe patents held by large existing companies. . . . Several of my interview subjects

4. For example, when I graded papers for Caltech's undergraduate intellectual property class, students would often post their work online and send me a web link. Some of these exercises could be found by search engines such as Google and were therefore distributed to the world. In a computer science class, this method of handing in homework could easily be the worldwide distribution of a patent-infringing program.

5. The low estimate is by Greg Aharonian (editor of the Internet Patent News Service), personal communication, July 23, 2004; the high estimate is from Bessen and Hunt (2004a).

6. Cherry (2004).

> joked that they thought it likely—without any investigation or particular knowledge—that there would be *something* in IBM's [patent] portfolio that their product infringed. . . .[7]
>
> Potential innovators know that the large mass of existing patents held by IBM and Microsoft are likely to receive some share of revenues from any major new product.[8]

The burgeoning number of multimillion-dollar software patent disputes shows that this concern is not merely speculative. Some major disputes are between well-known firms such as Adobe and Macromedia, Yahoo! and Google, or Id Software and Creative Labs. Many others have pitted small firms in the business of lawsuits against large software companies, as in the suits of Acacia against nine cable companies; American Video Graphics against twelve video game vendors; British Technology Group against Amazon.com, Microsoft, Apple, and vendors of virus-detection software; Eolas against Microsoft; and DE Technologies against Dell.[9] The list goes on. Note well that none of these suits allege that the defendant read the plaintiff's work and then appropriated it without permission; in every case two groups independently arrived at the same algorithm, and the one with the patent sued the other.

Whether these claims are justified or not, each funnels millions of dollars out of research and design of better software and into the legal system. If nothing else, this book proposes clarifications of the rules on software patents so that disputes either do not arise or are settled efficiently.

Free Software

Another noteworthy example is *Kodak* v. *Sun*. Sun developed the Java programming language (discussed further in chapter 7) and gives it away

7. Mann (2004, p. 53).

8. Mann (2004, p. 57).

9. On Acacia's suit, see p. 89; on AVG's suit, Fred Locklear, "Patent Aggregator Attempts to Make Tech and Game Giants Bleed," *Ars Technica,* November 5, 2004 (arstechnica.com/news.ars/post/20041105-4374.html); on *BTG* v. *Amazon,* Douglas Sorocco, "Amazon, Netflix, and Overstock Sued for Internet Visitor Tracking Patent Infringement," *PHOSITA,* September 17, 2004 (www.okpatents.com/phosita/archives/2004/09/amazon_netflix.html); on *BTG* v. *Apple* and *MSFT,* John Oates, "BTG Sues Apple and MS over Software Downloads," *The Register,* July 21, 2004 (www.theregister.co.uk/2004/07/21/btg_sues_apple_microsoft/); on BTG and virus detection, "UK Firm Patents Software Downloads," *The Register,* June 16, 2004 (www.theregister.co.uk/2004/06/16/uk_

for free, in the hopes that it will expand the company's hardware sales. Kodak proved to a court that Sun was infringing on a handful of patents that Kodak had bought from Wang Laboratories (now Unisys) and then settled with Sun for $92 million dollars in damages—from *free* software.[10]

Although free and open software has become an increasing part of the business strategies of many companies and even of governments from Munich to Delaware to Venezuela, Kodak has proved that any such decision brings a liability risk.[11] One group found that the Linux kernel, the most high profile piece of open software and one of the most widely used, potentially infringes 283 patents.[12] Under such circumstances, businesses and governments may be reluctant to take advantage of the public good that Linux's developers have created—the city of Munich has already put a brief delay in its Linux migration plans because of concerns about fifty patents on inventions such as browsers that allow navigation with the <tab> key (see figure 2-4).[13] In the United States, the Department of Defense, Census Bureau, and National Aeronautics and Space Administration are all involved in open-source projects, saving taxpayers money over proprietary alternatives—but what happens if a patent-holder sues the Department of Defense for infringement?[14] The potential liability from free software written in-house or by others could cost taxpayers even more than the $92 million Sun paid out.

---

firm_patents_downloads/); on *Eolas* v. *MSFT,* p. 86; and on *DE Technologies* v. *Dell,* Tony Smith, "Dell Sued for Alleged Global Sales Patent Abuse," *The Register*, November 5, 2004 (www.theregister.co.uk/2004/11/05/dell_e-commerce_patent_clash/).

10. Ashlee Vance, "Sun Settles Java Spat with Kodak for $92 Million," *The Register*, October 7, 2004 (www.theregister.co.uk/2004/10/07/kodak_sun_settle/). See also the presettlement report, John Oates, "Kodak Wins Sun Java Patents Case, Wants $1bn," *The Register,* October 4, 2004 (www.theregister.co.uk/2004/10/04/kodak_wins_java/).

11. Other countries that mandate the use of open-source software in government include Argentina, Brazil, Bulgaria, Chile, Colombia, France, Italy, and Peru. Countries that have a stated "preference" for open source include Bahrain, Belgium, China and Hong Kong, Costa Rica, France, Germany, Iceland, Israel, Italy, Malaysia, Poland, Portugal, Philippines, and South Africa. Robin Bloor, "The Government Open Source Dynamic," *The Register,* January 7, 2005 (www.theregister.co.uk/2005/01/07/gov_open_source_dynamic/). On Delaware and Munich, see Galli (2003).

12. Stephen Shankland, "Group: Linux Potentially Infringes 283 Patents," *CNET News.com,* August 1, 2004 (news.zdnet.com/2100-3513_22-5291403.html). Notice that the study is by a firm that sells IP lawsuit insurance, so the number is likely to be biased upward. Nonetheless, the exact number is not important: the entire project could conceivably be shut down by one or two key patents.

13. Patentrecherche Linux-Basisclient München (www.presseportal.de/showbin.htx?id=31139&type=document [German pdf]).

14. Galli (2003).

Copyright

The correct breadth of a patent, in the legal and economic sense, covers the details of an idea's implementation, not the broad idea itself. For software, that means lines of text. Copyright, which also protects text, has a few major advantages over patents, notably regarding independent authorship.

If users cut and paste another person's code into their own without permission, that act is a clear-cut copyright violation. But what if two people independently write the same code? A thousand monkeys with typewriters would need a thousand years to hammer out an exact copy of *Hamlet*, but if two programmers needing a pop-up window both wrote code exactly matching that in figure 1-1, it would be no surprise at all. In the patent world, every such coincidence is a lawsuit in the making; in a copyright regime, multiple inventors will not be able to harass each other, because independent authorship is indeed a valid defense for copyright cases.

On the other hand, whereas two bodies of code that look alike may have been independently invented, a body of code that looks nothing like another may be a direct plagiarism with a trivial translation. There needs to be a mechanism in place to facilitate verification of independent invention, which can be done via inspection of the process by which a given program has been written. The details of how copyright should be applied to code are discussed in chapter 8.

Politics

The primary policy recommendation of this book is that the U.S. Congress needs to consider what sort of IP protection is appropriate for software. To date, the law governing software has been entirely written by the courts, which do not have the authority to settle policy questions, only to interpret the intent of Congress, as made clear in the dissenting opinion in *Diamond* v. *Diehr* (discussed in chapter 4): "The broad question whether computer programs should be given patent protection involves policy considerations that this Court [the Supreme Court] is not authorized to address." The Congress therefore needs to decide the optimal policy for software. I hope that this book will provide a good start for the debate.

The European Union recently concluded a heated battle over the patentability of "computer-implemented inventions" in its legislature. After years of fierce debates and protests, no law of any sort was passed; the parties are now preparing for the next round. Unfortunately, I cannot include discussion about the battle, because political events move so quickly that whatever I write will not be current by the time this book reaches print. Instead, I have focused on the more universal topics of patent policy from a mathematical and economic perspective. The case law in chapter 4 is U.S.-specific, but the European Patent Office accepts the same "general-purpose computer with software" wording trick that I discuss extensively and faces the same fundamental questions about the patentability of mathematics and software. Thus although I will not explicitly discuss the European debate, this book has immediate relevance to it.

Another battleground is in trade-related intellectual property (TRIP), and here too I avoid the ensuing international relations issues, which pertain largely to the harmonization of laws in different countries. As a practical matter, this means persuading others that they need to adopt U.S. IP law as their own, so if the United States adopts a patent policy that is ill conceived, trade negotiations may spread this policy the world over. Again, the politics of TRIP negotiation is ever changing, but the economics of good patent policy as discussed in this book is not.

About the Book

A proper discussion of software IP will gather together perspectives from law, computer science, mathematics, and economics. This book is intended to provide a discussion of software intellectual property in all of its relevant contexts.

The first considered here is the economic perspective. Chapter 2 opens with an overview of patents and copyrights and then turns to the most important economic question about their design: how broad should protection be? That is, should an innovator have protection only from direct plagiarism, or from more loose imitation?

Focusing on the computer science side, chapter 3 examines the structure of software—the layers upon layers of complexity that have evolved to make it easier and easier for programmers to write useful programs. This chapter also introduces the mathematical context, explaining the extent to which mathematics and computer science overlap.

Chapter 4 explores the legal context through a history of how the courts have dealt with software. As in the mathematical context, software falls somewhere along a continuous spectrum of invention between physical machines, which should be patentable, and pure math, which should not be. Exactly where should the legal line between the patentable and the unpatentable be drawn? Since judges are knowledgeable about law but light on computer science, it is no surprise that the line drawn by the courts has proved to be in entirely the wrong place.

Chapter 5 elaborates on this question in the business context. Now that we have a firm rule about what may be patented, what are its effects on business and innovation in the real world? Is there evidence that it has led to more innovation than it has stifled? (Hint: no.)

Among the producers of patented goods, the software industry is unique in an interesting way: it is rather evenly split between those who make money by writing software for hire (that is, a labor market) and those who make money selling software via shrink-wrapped CDs (that is, a goods market). Patents do not affect both sides of the market equally. Chapter 6 discusses the bifurcated market and how patents shift the balance between the two sides.

The music and movie industries offer another important perspective on software, covered in chapter 7. The debate over what constitutes fair use of media is not one that I touch upon here, but that debate has spilled over into important issues of software protection. If a music label invents a special encoding for its music and distributes a program with which to play it, do users have the right to write their own software that can decode the music without the permission of the label? The current answer, according to the U.S. Congress, is that users do not have such a right. But since encryption and copy protection are so hard to define and delimit, this rule has turned into the broadest possible form of IP protection: once a software author has claimed that his or her work implements a copy protection scheme, that author can claim exclusive ownership of the right to produce any of a variety of add-ons, extensions, and accessories. No software is an island, entire of itself, so the ability to block competitors from producing interoperable software is an immense power that can be readily abused. Such abuses have already appeared in the courts, creating still more IP headaches for anyone who writes software, be it for music, electronic books, or garage-door openers.

Chapter 8 considers software as a literary work—*Hamlet* with numbers. As already mentioned, the process of writing a play and the process

of writing a program are obviously very different, and IP rules need to take that into account. For those who have read this far and still believe that patents are appropriate, I offer suggestions on how patents can accommodate inventions-in-words. The natural protection for words on paper is copyright, but this too is not quite a perfect fit, so I also discuss how copyright for software could be more effectively implemented.

Chapter 9 collects and summarizes the policy recommendations from all of these perspectives.