Innovation, IP, and the Protection of Software

Mike Kazanjy, Sam Lester, Brian Leubitz, Andrew Pardoe, and Tao Yue

Software has fallen through the cracks of the modern intellectual property (IP) system since the dawn of computing. Lawyers and judges were not comfortable with what software really is and how it fits into old IP paradigms. This paper will begin with a review of the history of IP and the problems of fitting old IP systems to new technologies. We will then look at why copyright has been selected and how it has been used in the past. We will then pause to consider the economics of innovation and some possible alternatives. Finally, we will look at two major cases, a series of "look and feel" cases and the case of Microsoft and Sun with the development of Java and Dot Net. In the end, it is clear that we need a system-wide review of IP legislation that protects software in more practical ways.

A Brief History of IP, and its Modern Day Failures

Intellectual property was an unknown concept before the mid 1400s. After all, there was no particular need for the ability to block the copying of artistic works in Europe until the Gutenberg printing press. Additionally, the difficulties surrounding travel until the dawn of the Renaissance slowed the spread of ideas sufficiently, such that patenting was not particularly worthwhile. However, improved technology in travel and printing during the Renaissance encouraged the concept of the protection of ideas and expression. This section will review the basic long-term history of intellectual property (IP) with an eye toward protecting software.

Patents

The concept of a limited monopoly based on a novel idea dates back to the Greek and Roman eras. The Greek historian Phylarchus recounted the story of the town of Sybaris, which granted one year monopolies to chefs who prepared particularly good dishes.[1] The Romans initially granted some limited monopolies, but ultimately grew disdainful of the concept of monopolies in general.[2]

However, trade began to surge after the French-based Norman Conquest. Soon, the French and English had built an extensive trading network leading to a greater need for protection of ideas. The first statute regarding patents was a Venetian statute in 1497 that granted ten year limited monopolies in exchange for disclosure of the idea to the Court.[3] In the 18th century, the British gave

---

1 Foster, Frank H. and Shook, Robert L, *Patents, Copyrights, and Trademarks*, 1993 at page 3

2 *Id.* at 4

3 "History of Patents", *Wikipedia*, *available at* http://en.wikipedia.org/wiki/History_of_patent_law

patent law its last major element, the requirement of written description.[4] At that point, the patent trade-off was clear: public disclosure in return for a limited monopoly. Since then, several new requirements and elements of a patent have been added, but the basics of patent law have remained relatively consistent. The question remains whether this centuries old system is still the best system to optimize the level of innovation.

The modern requirements to attain a patent are essentially the following:

1. The inventor discloses his invention, including the "best mode" of recreating the invention.
2. The inventor clearly claims his invention.
3. The claims of the invention are new and not obvious to one who is "skilled in the art."

These rules have been similar for the past two hundred years with only minor modifications. Until recently, the software patent was an unknown concept, but has now grown by leaps and bounds. While the prosecution of software patents is particularly onerous, especially when a software patent application is deemed a "business method," the use of the patent system will likely continue to be a tool of increasing importance in the protection of software.

Copyrights

Unsurprisingly, the first inklings of copyright in the Western world occurred slightly after the creation of the printing press, but it did not truly develop in Europe until the literacy rate improved in the following centuries. However, during the one hundred years following the invention of the printing press, the European Crowns granted "special privileges" that looked remarkably like modern-day copyright.[5]

In England, the King, by fiat, enacted the Licensing Act of 1662, which established a registry for publications. However, the Stationers' Company, a powerful guild at the time, quickly reclaimed this power. The Stationers' rules required their members, who owned virtually every printing press in Britain at the time, only to print works which they had permission to print under the scheme. At this point, the printer, not the author, owned the right to distribute the work.[6] As the power of the Guilds began to wane, Parliament passed the Statute of Anne, the first comprehensive European copyright law, in 1709.

The Statute of Anne established a few crucial points. Most importantly, it established the author,

---

4 "Five Hundred Years of Patents: The Eighteenth and Nineteenth Century", British Patent Office (archived site), *available at* http://web.archive.org/web/20060505210345/www.patent.gov.uk/patent/whatis/fivehundred/eighteenth.htm

5 "History of copyrights", *Wikipedia*, *available at* http://en.wikipedia.org/wiki/History_of_copyright_law

6 "Copyright History," British Patent Office (archive), *available at* http://web.archive.org/web/20060502234005/http://www.patent.gov.uk/copy/history/index.htm

rather than the publisher, as the owner of the copyright. While the importance of this change is today diminished by corporate ownership of works for hire, it remains relevant for small or solo developers. Additionally, the Statute of Anne extended copyright to other media.

The Berne Convention, which was negotiated in 1886, was an international effort at standardization of copyright regulations. Initially, it was a bilateral agreement between the United Kingdom and the United States, but has since expanded either through the WTO or by direct ratification to almost every nation.

Copyright today is an easily obtainable protection, in that no process is really required. To obtain a copyright, the author must only capture the work in a "tangible medium," whether that be a computer disk, a piece of paper, or other recording. At the moment of expression, copyright vests in the work. However, copyright protects only the expression of an idea, not the actual idea. This has profound implications, especially in the area of look and feel for software. This concept will be explored further in the paper.

Trade Secrets

Trade secrets are the oldest of intellectual property protections. Trade secrets have been used since time immemorial to protect inventions. By hiding the invention or process, the inventor can stop other producers from making the invention. However, trade secrets do not stop competitors from reverse engineering. So, for many products which are easily reverse engineered, trade secrets have no application.

Trade secrets are protected by the government only to the extent that the law forbids industrial espionage. Thus, the strength of trade secrets is very limited and requires a high level of security to avoid disclosure. In the world of modern day software and the decompiler, trade secrets have limited applicability.

Timing Problems

In the United States, patents are generally granted for a term of twenty years from date of filing, but until recently the term was seventeen years from date of issuance. Such long patent periods have become unnecessary in many modern high-tech applications. In fact, many patent applications are stuck in the patent office for longer than their useful life. With increasingly rapid technological innovation, the question must be raised: Is our patent system falling behind?

In FY 2006, the United States Patent Office received over 417,000 Utility, Plant, and Reissue patent applications, and over 75,000 additional international and design applications. Current average

processing time is over 31 months, or slightly more than 2.5 years![7] By the time many of these patents are settled, they are no longer useful. This is particularly true for software patents due to the long prosecution times, thus software companies have been reluctant to rely solely on patents. Further, the patent process has, in many ways, become nothing more than a numbers game. Patent practitioners are given an incentive to produce a large quantity of patents and claims. By the time many patents are issued, they are not even in use. So they become mere fodder for an arms race between massive corporations and are otherwise useless. It's not clear that this protects anything, except perhaps the ability of patent practitioners to bill their clients.

Copyrights, on the other hand, have no formal process and are more adaptable. However, they do not carry the same level of protection. Does copyright provide sufficient protection to ensure that the R&D work by software companies will not be stolen? Can software companies rely on trade secrets at all? Given the power of modern decompilers, where does reverse engineering end and industrial espionage begin?

This paper will attempt to outline some uses of intellectual property during recent technological innovation. Further, we will discuss some legal cases in order to tease out how intellectual property can serve to further innovation in modern computing.

Legal History of Software Copyright

Copyright Legislation

In the United States, copyright in software derives from the Copyright Act of 1976. Despite being a complete overhaul of the nation's copyright regime to reflect seven decades of technological change, the Act referenced software only peripherally. The act merely "provided that computer programs should have no greater protection than they had enjoyed under prior law," despite the fact that the earlier 1909 copyright left the status of software "completely unclear."[8] Two years earlier, Congress had tasked the National Commission on New Technological Uses of Copyrighted Works (CONTU) to investigate software IP. The Commission took four years to recommend that software be placed on equal footing with other forms of copyrightable materials, and Congress took another two years to codify those policies into the Copyright Act of 1980.

This episode is indicative of the difficult fit between software and IP law. A fast-moving field such

---

7 "FY 2006 Annual Report", United States Patent and Trademark Office, *available at* http://www.uspto.gov/web/offices/com/annual/

8 Clapes, Anthony Lawrence. *Software, Copyright, and Competition: The "Look and Feel" of the Law.* Westport, Connecticut: Quorum Books, 1989, at p. 15

as computing will necessarily develop faster than the law can anticipate implications. In a mere 11 years, text-based MS-DOS (1981) was largely supplanted by the graphical Windows 3.1 (1992). Yet it took six years to resolve the copyrightability of software. Indeed, copyright legislation has a history of being reactive, rather than prospective. The 1909 Act failed to include motion pictures, which had already been a popular form of entertainment for a decade. Sound recordings, an even older form of media, were not made copyrightable until 1971, when widespread availability of transistorized cassette tape recorders had made "piracy of sound recordings … a problem of frightening proportions."[9]

Instead, the principles of software copyright have been left largely to the judicial system to define. Much of the reasoning is found in lengthy court decisions, which try to make tradeoffs between the public good and the rights of IP holders. In contrast, Congressional committee hearings usually come across as televised theater, replete with politicians' favorite sound bites about the number of job losses prevented[10].

Protecting Software against Direct Copying

Since software is treated on par with books, music, film, and other literary works in its eligibility for copyright, it is clear that direct copying would amount to piracy. But software can take more forms than most literary works. Source code is what the programmer writes, and would be analogous to a book manuscript. But object code is understandable only by the computer. Is it protected? What about object code burned into firmware, which is not fixed in an easily-readable medium?

Such questions were raised in the seminal case for software copyright: *Apple* v. *Franklin*. At this point in early personal computing, much computer functionality was shipped out on firmware. This was true of the Apple operating system, whose chips were copied for Franklin's clone of the Apple II. Letting himself be swayed by the literary analogy of software, Judge Newcomer determined that object code was not copyrightable, positing that, "If the concept of 'language' means anything, it means an ability to create human interaction … To go beyond the bounds of this protection would be … [to] step into the world of Gulliver where horses are 'human' because they speak a language that sounds remarkably like the one humans use."[11] Furthermore, software in ROM "seemed to be physical devices … [which] copyright does not protect".[12]

---

9 *Id*, at p.15

10 *Criminal sanctions for violations of software copyright: hearing before the Subcommittee on Intellectual Property and Judicial Administration.* 102nd Congress, 2nd Session. GPO, 1993. Or see any number of other such hearings before the House or Senate subcommittees.

11 545 F. Supp. 812, E.D. Pennsylvania 1982

12 Clapes, at p. 32

The 3rd Circuit Court of Appeals took a broader view of software, noting that software by definition could be run on a computer to produce a desired effect. Thus, object code is just as much "software," and just as protected, as source code. Likewise, it does not matter in what medium the software is located; the software itself is the protected entity, and putting it in a chip doesn't automatically turn it into a "device". Franklin had also advanced the argument that an operating system, by its very nature, could only be implemented in a certain way, i.e. that in the OS, idea and expression had merged. Copyright does not protect ideas, but rather the expression of an idea. If there is only one way to express an idea, then it is not copyrightable. The appeals court did not address this issue directly, finding the trial record insufficiently clear on this question, but the merger of ideas and expression would become a recurring feature in future court cases.

Other court cases rapidly settled the copyrightability of software in any form. In *SAS* v. *S&H,* S&H was found to have appropriated, and rearranged, the source code to the SAS statistical package in an attempt to port the program from the IBM System/370 to the DEC. Like an author caught with snippets from another author's book, S&H argued that there weren't enough similarities to justify the charge. The court disagreed, finding that "the taking of even a quantitatively small fragment … might be qualitatively substantial because of [its] disproportionately large value …"[13] In *NEC* v. *Intel*[14] and *Allen Myland* v. *IBM*[15], micro codes in CPUs were also held to be copyrightable. This was not a great leap from firmware, but micro codes do operate at a lower level in the computer system, closer to the hardware. Direct copying infringes copyright no matter how much is done and what form it takes.

What Else is Protected in Software?

The issue of idea vs. expression came up more directly in several cases where code was not copied directly. In *Whelan* v. *Jaslow*[16], Whelan had written a dental laboratory administration package in the EDL programming language for Jaslow Labs. Later, after the two had parted ways and Whelan had begun selling the program to other dental labs, Jaslow tried to reimplement the program line-for-line in BASIC and sell it in competition with the original Dentalab program.

Did the code rewrite infringe Whelan's copyright? BASIC and EDL are different languages, and compile to different object code even if a program is ported line-by-line. The court found, and the

---

13 605 F.Supp. 816 (M.D. Tenn. 1985)

14 10 U.S.P.Q.2d (BNA) 1177 (N.D. Cal. 1989)

15 33 F.3d 194 (3d Cir. 1994)

16 797 F.2d 1222 (3rd Cir. 1986), cert. denied, 479 U.S. 1031 (1987)

appeals court affirmed, that the "structure, sequence, and organization"[17] of the program were protected as expression. The idea was that of a dental program; anyone could implement another dental program, but not by imitating the structure of Dentalab. Widely criticized as the time for being overly broad in its definition of expression (how many ways are there, really, of structuring a dental program?), this decision is considered "the high-water mark for copyright protection,"[18] and "the most extensive scope of protection U.S. courts had *ever* accepted." [19]

Further court decisions were more circumspect in their wording, but the general principle holds. Clearly, more was protected in a software program than the exact duplication of parts of the program. In *Broderbund* v. *Unison World*,[20] menu structure was found to be protected by copyright. Broderbund's Print Shop was an Apple II program for printing greeting cards and other artwork projects, showing so much promise that Unison expressed interest in porting it to the IBM PC while the program was still in development. The deal with Broderbund later fell through, but Unison modified the port and released it anyway. In court, Unison argued that *any* menu-driven program designed for printing such computerized projects would have to be substantially similar to the Print Shop program. Unfortunately for Unison, Broderbund located "a program with the cheerful name Stickybear Printer" which did much the same thing as Print Shop, but without such extensive similarities. Unison had also been so faithful to the original Apple II program that users were instructed to press "Return," despite the fact that the IBM PC keyboard called the key "Enter."[21]

The *Broderbund* case was one of the first to touch on look-and-feel, leading to the various *Lotus* cases, and the numerous *Apple* cases. Look-and-feel is sufficiently complicated that it will be explored in more detail in a section later in this paper.

Reverse-Engineering

The rationale for time-limited exclusivity in intellectual property rights is, per Article I, Section 8, Clause 8, of the US Constitution, "to *promote* the Progress of Science and useful Arts" (italics emphasis mine, uneven capitalization James Madison's). In other words, we would not have IP rights if not for the encouragement it gave authors and inventors to create works for the benefit of others. This is historically different from the European approach, which "emphasize[s] the

---

17 Drexl, Josef. *What is Protected in a Computer Program? Copyright Protection in the United States and Europe.* IIC Studies in Industrial Property and Copyright Law Vol. 15. Weinheim, Germany: VCH Publishers (VCH Verlagsgesellschaft), 1994, at p.20

18 Graham, Lawrence D. *Legal Battles that Shaped the Computer Industry*. Westport, Connecticut: Quorum Books, 1999, at p. 88

19 Drexel, at p. 18

20 648 F. Supp. 1127 (N.D. Cal. 1986)

21 Clapes, at pp. 141-151

protection of authors … The German Copyright Act, as an example … starts with the author and only in section 2 defines the concept of copyrightable works."[22]

But American and European approaches have been converging, especially regarding reverse engineering. The *idea* behind a copyrighted work is free for anyone else to implement, so long as it is done in a unique manner from the original implementation ("expression" in the copyright lingo). But the source code to computer programs is usually kept secret. Thus a limited degree of reverse engineering is sanctioned under the law, so that ideas are not also kept out of the public domain. Taking a different tack, Europe "justifies the legality of reverse engineering … on the grounds of free competition by *interoperability of systems*."[23] (Italics mine; shades of the recent EU fight with Microsoft over protocols!)

In order to separate the unprotected idea from the protected implementation, a "clean room" technique is typically used. A group of engineers is tasked to analyze and document the functions of a competitor's product, but a *different* group of engineers, uncontaminated by knowledge of the competitor's implementation, designs their own implementation. Communication between the two groups is documented minutely and passed through the lawyers, to provide a paper trail showing that design decisions were made organically based only on the specification. The burden of proof lies with the second implementer. So even "documentation of all the mistakes made along the way … [hopefully] will prove that the programmers' work is original."[24] One of the most famous examples of clean-room engineering is the cloning of the IBM PC BIOS by Compaq.[25] Likewise, in *NEC* v. *Intel*, NEC cleverly established the originality of its implementation by commissioning a clean-room copy. The judge "found the results of this exercise profound",[26] and took note of the fact that the code in question had as many similarities with the clean room code as with the Intel code.

Perhaps the most direct validation of clean-room techniques is *Computer Associates* v. *Altai*.[27] Some CA employees had jumped ship to Altai to work on a job scheduling program. Altai released its OSCAR 3.4 program, which used about 30% of CA's ADAPTER code directly, but clean-roomed the offending code out of OSCAR 3.5. The district court found, and the Circuit Court affirmed, that version 3.4 was infringing but 3.5 was acceptable. To reach this conclusion, the circuit court further chipped away at the broad *Whelan* protection of program structure, considering similarities at various levels of abstraction and determining which elements were dictated by external or

---

22 Drexl, at p.1
23 *Id.* at 12
24 Clapes, at p. 153
25 Cringely, Robert X. *Accidental Empires*. Collins, 1996, at pp. 170-173
26 Clapes, at p. 129
27 775 F.Supp.544, 549-55 [20 USPQ2d 1641] (E.D.N.Y. 1991)

efficiency factors. Similarities, then, can be justified, even when created for the express purpose of compatibility. Certainly this is a more complicated test, but a narrower and much less controversial one.[28]

However, reverse engineering remains fraught with perils. Competitors must strictly follow the guidelines to be assured of legality. In *Johnson* v. *Uniden* [29], Uniden's compatible radios were found to have infringed on the software contained in Johnson's originals. Uniden argued that they had copied only those segments necessary to ensure compatibility. However, several telltale signs, including three lines of nonfunctional code found in both implementations, convinced the court that copying went beyond what was strictly necessary.

Even when following the guidelines, reverse engineering is rife with hazards. An example comes from the video-game market. Sega took technological measures with its Genesis III console to lock out games from non-Sega licensees. The console looked for the letters "S-E-G-A" in the game program, and used that to check initialization code. The hope was that the Sega trademark, and the fact that a Sega copyright screen came up when the initialization code was detected, would prevent the production of games using reverse-engineering methods. In *Sega* v. *Accolade*[30], the trial court prohibited Accolade from decompiling the Sega program code, and required the recall of all offending games within ten days. Sega also submitted examples of cartridges which didn't use Sega's initialization code, and which ran on the Genesis III, but "refused to allow Accolade's software engineers to examine them or to reveal the manner in which they had been modified." It is thus unclear if a legitimate reverse-engineering method would have discovered such noninfringing methods. In a narrow escape on emergency appeal, the circuit court reversed the ruling, finding that disassembly was fair use since it was the "'only way to gain access' to the functional elements contained within the software."[31]

Direct Copying, Redux – Consumers

While the courts occupied themselves with companies copying each others' products, the Congress was busy satisfying industry lobbyists. The wealth of prepackaged software for personal computers in the 1980s led to the formation of a cottage industry for software rental. Little more than a nudge and wink was given to software piracy. One might watch a video and take it back to the rental store, and the natural analog degradation of videotape copies placed some limits on consumer copying. But was anyone really renting a copy of Windows 2.03 or WordPerfect 5.0 to use for a week? Even large software retailers like Egghead got into the act, charging substantial

---

28 Drexl, at pp. 23-27

29 653 F. Supp. 1485 (D. Minn. 1985)

30 977 F2d 1510 (9th Circuit 1992)

31 Graham, at p. 116

restocking fees for returned software in opened boxes.[32]

Faced with a challenge to its campaign donors, the Senate Subcommittee on Patents, Copyrights, and Trademarks held hearings on the matter. It was a fun bit of televised theater where the President of Microsoft extolled his company's contribution towards reducing the American trade deficit with Japan, and Senators reminisced about a certain brilliant young man (Bill Gates) who'd once been a Congressional page[33]. Congress quickly passed the Computer Software Rental Amendments Act of 1990, prohibiting "rental, lease, or lending" of computer software and sound recordings. Evidently the motion picture industry had spent all its political capital with the Betamax fight, since videos remained available for rental.

But not to worry, Sonny Bono's tragic death led to the Copyright Extension Act of 1998, extending copyright for an additional twenty years for existing works. The move towards digital video and the possibility of making perfect copies was countered by the Digital Millennium Copyright Act (DMCA, sung to the tune of YMCA) of 1998[34], which among other provisions prohibited the circumvention of access controls for digital media.

Although intended to protect digital music, video, and other forms of video, the broad wording of the anti-contravention clause of the DMCA has led to unintended side effects in unrelated areas. Can researchers circumvent access controls on music?[35] On eBooks?[36] Can companies build garage door openers to compete with ones that have protected interfaces?[37] Or inkjet cartridges?[38] Is reverse-engineering being chipped away by the creative use of the DMCA's anti-circumvention clause?

Why Copyright?

The DMCA question leads into the question of whether copyright is appropriate. All of the aforementioned DMCA cases were eventually resolved in favor of the reverse-engineers. *Felten* was

---

32 United States Congress. Senate Subcommittee Hearing 101-456. Government Printing Office, 1990

33 *Id.*

34 17 USC section 1201a

35 See the Electronic Fronteir Foundation's information site for Felten v. RIAA at http://www.eff.org/IP/DMCA/Felten_v_RIAA/20020206_eff_felten_pr.html

36 Dmitry Sklyarov was arrested while attending a conference in the United States, since he had worked on breaking Adobe's eBook access controls.

37 *Chamberlain v. Skylink*, 381 F.3d 1178 (Fed. Cir. 2004)

38 Lexmark v. SCC, 253 F. Supp. 2d 943 (E.D. Ky. 2003)

mooted because the RIAA did not in fact succeed in preventing the publication of the SDMI paper. Sklyarov was released from jail, and his employer ElcomSoft was acquitted in a jury trial. Chamberlink's claim against Skylink was rejected, and the appeals court noted that "the DMCA … does [not] … divest the public of the property rights that the Copyright Act has long granted to the public.".[39] Lexmark also lost its case. Outside of the court system, there have also been some victories for reverse-engineering. Very recently, the Librarian of Congress issued exemptions to the anti-circumvention clause, including "sound recordings … protected by … measures that … create or exploit security flaws or vulnerabilities", a clear reference to the Sony rootkit scandal.[40]

Taken from a historical perspective, this situation appears to be not much different from prior cases. There is a balance between the broad and the narrow-constructionists, between the more-protection and the more-commons sides. *Whelan* found program structure to be protected by copyright, whereas *Computer Associates* narrowed the protection to just those program elements which are not constrained by permitted factors. Decompilation is permitted, but you cannot just take the results and put it into a new program. Two steps forward, one step back. Some cases were poorly and/or strangely decided, as was *Sega* originally. But the appeals court saved the day for reverse engineering. So long as the resulting decisions seem reasonable to an observer interested in ensuring protection while promoting technological progress, it cannot be said without caveat that the system is entirely broken.

Doubtlessly there are some cases where the DMCA, or other copyright laws have been poorly applied. But on the whole, software copyright's broad principles are relatively settled law. You cannot copy programs directly. You can copy programs indirectly, but within bounds, and reverse-engineering is permitted to a point. Companies can launch lawsuits under the DMCA, as they can under any law, but if the facts and precedent do not support their case, they will lose. Private suits, including frivolous ones, are a fact of life in our system. With new laws such as the Copyright Act of 1976 (amended 1980 for computer software), it took over a decade to hash out the details of ideas, expression, and look-and-feel. The same appears to be happening with the DMCA.

Why copyright? Because it already exists. Because its "adaptability … has meant that the risk-takers … could rely on a well-understood, long-standing body of law to protect their investments."[41] On the other hand, just because copyright law is settled does not mean we shouldn't question its effects. Professor Maurer raised the question of why source code is kept secret when copyright law is supposedly so strong. In other words, why not publish it if you know people will be afraid to use

---

39 *Id.*

40 Rulemaking on Exemptions from Prohibition on Circumvention of Technological Measures that Control Access to Copyrighted Works. http://www.copyright.gov/1201/

41 Clapes, at p. 16

it? Well, it's in the company's own interests to keep it secret. Source code "transfer[s] ideas to a skilled person."[42] Companies wish to retain all advantages possible, so they would like to preserve the secrecy of code at the same time that they have copyright. The law does not require them to choose one or the other, so they choose both. Would it be sufficient to protect object code with copyright, and source code with trade secret? Perhaps technology has taken that solution away from policymakers. With the advent of managed-code, Java, and .NET, decompilation now results in much more readable code than in the native-code era. The difference between source and object code is diminishing.

But one clear theme is that Congress seems to have largely delegated its responsibilities to the courts. With the initial development of copyright law, "Congress offered little guidance to the courts … With each lawsuit, the courts added to [its] … sketchy foundations."[43] The 1976 Copyright Act took 21 years to complete.[44] The DMCA may have caused fewer attempted extensions if it had been more narrowly written. This situation is not limited to copyright. With patents , too, "Congress, despite anxious pleas from the Supreme Court to solve the issue by statute, [did] nothing."[45] Among both supporters and opponents of copyright for software, there is a consensus that the legislature has been sitting on the sidelines. As President Warren G. Harding said, perhaps apocryphally, "The business of government is business." It seems, for the legislative branch at least, this business does not include lawmaking in the area of copyright law.

The Effect of IP on Innovation in Software, and Alternatives

Innovation is a phenomenon that is mysterious and hotly debated. The economics of innovation force tradeoffs between encouraging innovation with intellectual property rights and creating deadweight loss with the monopolistic behavior encouraged by those very rights. While intellectual property rights are said to promote innovation by providing incentives for inventors to come up with new ideas, they also encourage individuals or teams to keep their ideas secret from other groups for fear of being "beaten to the punch." This lack of collaboration between racing groups can lead to duplicated efforts that waste resources. The complicated economics of innovation have led to various alternative methods of encouraging innovation, including direct patron-client relationships and other collaborative efforts. In the computing industry, innovation, incentives, and intellectual property have been especially complicated, as information sharing has developed. This

---

42 Drexl, at p. 11

43 *Id.* at p. xi

44 Borking, John J. *Third Party Protection of Software and Firmware*. Amsterdam: Elsevier, 1985, at p. 218

45 *Id.*

part of the paper will discuss the economic rationale for intellectual property and its design, its common downfalls, and alternatives in public support and the open source movement.

Economics of Innovation

Today's "information goods," such as software, digital music and other forms of digital entertainment goods, meet the same criteria as a public good. Paul Samuelson is commonly identified as the first to identify public goods, defining the public good to be one "which all enjoy in common in the sense that each individual's consumption of such good leads to no subtractions from any other individual's consumption of that good…"[46] This is the "non-rival" characteristic of public goods. Public goods are also defined to be "non-excludable," meaning that no person can be excluded from consuming public goods. Information goods, and knowledge goods in general fit the non-rival part of the definition and, hardware barriers to entry aside, non-excludability has also developed as transfer rates increase. Since information goods act as public goods in the economy, knowledge markets suffer from the common market failure of under -provisioning of public goods. There are various methods of addressing this market failure and the most appropriate method most commonly depends on the market in question.

Intellectual property rights have become a common incentive for innovation. By offering exclusive rights to a bit of knowledge, a certain production process, or a template for production, intellectual property rights decentralize decision-making for innovative processes.[47] The promise of exclusive rights allows inventors to make their own cost-benefit analysis of investing time into the development of a product. This encourages inventors to make decisions based on the most accurate projections they have available. Ex ante analysis of intellectual property is efficient, providing innovations that would otherwise not exist. Ex post, however, intellectual property rights result in deadweight loss, because the marginal cost of knowledge is effectively zero.

According to Scotchmer, all branches of intellectual property have features in common, including "a stipulated duration, a notion of 'breadth' (usually not established by statute, but rather in case law), a notion of minimum protectable inventive step, and some 'fair-use' exemptions (with the exception of patents)."[48] These features allow policymakers to adjust the strength of intellectual property rights, accordingly with the characteristics of the specific innovation. The social value of an invention decreases as the duration of protection increases because the deadweight loss associated with protection is greater for longer durations. This implies that the efficient duration of protection would be just long enough for the innovation to be made.

---

46 Paul A. Samuelson (1954). "The Pure Theory of Public Expenditure". Review of Economics and Statistics 36 (4): 387-389

47 Scotchmer p.97

48 Scotchmer p.97

Another potential social loss associated with strong intellectual protection is duplicated costs that arise when more than one group conducts research to a common objective. Assuming that ideas are not scarce, several groups or individuals will often pursue an innovation, separately, in a so-called "race." Duplicated costs are not always the outcome of races, especially if the probability of success in a research operation is not 100%. This allows a concept of an optimal number of participants in such a race, depending on the probability of success, the value of the intellectual property right and the costs of research.[49]

Patent races, as an example, often involve several firms or several groups within a single firm competing for a similar functional objective. This has historically led to duplication of research costs amongst the competing groups. Zeira suggests that "risk aversion can lead to cooperation between participants of a patent race, since by sharing the gains from innovation they keep the same expected income but reduce risk."[50] Zeira further suggests that this behavior is much more common when patent races are internalized by a single firm, by hiring multiple groups to achieve the same objective. Collaboration between competing firms or groups begins to address the issue of duplicated costs by sharing knowledge and ultimately the benefits of research. While this behavior is not common in innovation markets, it suggests a solution to duplicating costs, by pushing on the risk aversion of separate firms. Aggregating this sort of collaboration over an entire industry would raise the base level of knowledge available to researchers, preventing social losses due to duplicated costs of research. When the probability of success for a particular objective is high, the optimal number of participants in a patent race is low, assuming they do not cooperate. However, a decentralized research effort that shares each step of the research could be efficient with many participants.

The cumulative nature of research raises another set of issues surrounding the protection of early innovators as newcomers commercialize and thus capitalize on the work of those who came before them. The Lotus/Borland copyright dispute over spreadsheet software, as well as Microsoft's dominance in both the office and operating system software markets, are examples of disputes over intellectual property rights in the computing industry where the cumulative nature of research in product development have come to the forefront. These cases are more specifically investigated in other parts of this paper.

Open Source - The Extreme Alternative

The open source movement developed, arguably, out of the 1960s counterculture movement, which

---

49 Scotchmer p.102

50 Zeira, *Innovations, Patent Races and Endogenous Growth*, p. 5

had a large impact on the development of the personal computing industry. John Markoff's book *What the Dormouse Said* outlines the influence of flower power on personal computing, making mention of open source proponents who played a key role in developing the industry. In his book, Markoff proposes that a large fraction of personal computing pioneers were essentially hackers, people who believed in free access to information and often lived by Proudhon's biting slogan "property is theft!"

The basic concept of open source as relevant computing is that software's source code (the information that makes up programs and makes programs work) be available to all for all bits of software. The economic rationale behind the open source movement is twofold: first, that by sharing information, access costs for consumers and creators are avoided; and secondly, that administrative and enforcement costs of intellectual property rights would be avoided. This implies that the duplicated costs of research in the pursuit of intellectual property would be avoided as well, assuming disclosure becomes commonplace under open source.

Some argue that open source would have a negative effect on innovation. Without the proper incentives to encourage development of new products, we cannot be sure that innovation would occur. In industries where research and development costs make up a huge portion of production costs, such as pharmaceuticals and aeronautics, open source may lack the necessary incentives for development of new products. However, with the large knowledge base that has been developed in the software and computing industries, open source has been quite successful.

Netscape's release of open source material under the name Mozilla has been successful in becoming one of the biggest providers of web browsing and email client software. The development of Wikipedia as an online resource has been a successful open source story. While the open source development of Linux has not taken on the market share of Microsoft's operating systems, Linux continues to grow its market share as an alternative for the more tech-savvy. For nearly all software applications, there is an open source alternative. It has largely been a matter of commercialization and advertising that has led companies like Microsoft and Apple to the forefront of the personal computing industry.

While there are clear deficiencies in the intellectual property model, open source is not a cure-all solution. The argument that open source lacks the incentive structure to be truly successful as a method of encouraging innovation holds strong, especially as the cost of research and development increases. Some also argue that much open source software is derivative, taking advantage of ideas whose development was initially paid for under the commercial model. Founders of the open source movement have aimed to downplay the adoption of open source in industries other than software, suggesting that weak arguments for open source in other industries simply weaken the

significantly strong argument for open source in software.

Explanation of Look & Feel

The term "look and feel" is used to describe the overall interaction between a user and a computer application. This includes the appearance (the "look"), comprised of colors, fonts, sizes, shapes, and layout, as well as the behavior of components, such as menus, buttons, and navigation (the "feel"). Companies attempt to maintain a consistent look and feel for several reasons. The first is that once the existing customer base is familiar with the functionality and navigation of a product, they are much more inclined to stay with the family of software instead of switching to a new product, with a new navigation structure and thus, a new learning curve to learn the new product. Another reason is for branding purposes. By maintaining a consistent look and feel throughout all of their products, a company creates brand value and recognition, allowing users to feel comfortable with any new product release since they recognize the product (or certain areas of the product) immediately.

Copyright of Look & Feel

Since the look and feel of an application is very hard to precisely define, it is natural that companies find it difficult to protect their innovations in this area, whether through patent, copyright, or other means. Over the past thirty years, there have been numerous cases of alleged infringement on various look and feel components.

One of the landmark cases that helped establish regulations surrounding look and feel protection came in 1990 when Lotus Development Corporation sued Borland Software Corporation over copyright infringement of their market-leading spreadsheet application, Lotus 1-2-3.[51] Lotus claimed that Borland had infringed on their spreadsheet menu structure, which they claimed was copyrighted intellectual property. They also claimed that Borland violated copyright by including the identical keystroke sequences used by Lotus 1-2-3. Borland's spreadsheet application, named Quattro, allowed users to switch to a "Lotus Emulation Interface", which allowed users to type the identical keystroke sequences used by Lotus 1-2-3. Borland claimed that even though the interface looked similar, the underlying code used in Quattro was completely created by them and was thus not an infringement. A district court ruled in favor of Lotus stating that Borland had infringed on copyright by using the identical menu structure. Borland quickly removed the Lotus-like menu structure, but retained their keystroke emulation interface, thus prompting another lawsuit from Lotus.

---

51 *Lotus* v. *Borland* 49 F.3d 807 (1st Cir. 1995) *available at:* http://www.law.cornell.edu/copyright/cases/49_F3d_807.htm

The result of the court's decision in favor of Lotus had significant impact on the world of computer software. As a result, companies could not release new products that strongly resembled existing products. This would in turn cause significant grief for end users since they would be forced to learn completely new systems when they transferred jobs or when their companies moved to new software applications. It also would encourage companies to be "first to market" since they could then in turn claim their stake in the market by releasing the product with the most intuitive and natural interface, blocking out competitors from releasing similar products. Similarly, this would cause problems for existing areas of software when multiple companies already had somewhat similar interfaces. Would these areas be forced to examine existing copyrights or would it be a matter of determining who created the interface first and in turn grandfathering that company with the rights to own the interface? This decision from the *Borland* v. *Lotus* case instantly shook up the software world and the means in which companies protect and release software, as well as stifling competition in certain aspects of software development.

Five years later, in 1995, the United States Court of Appeals for the First Circuit reversed the district court decision claiming that the navigation structure used by Borland was a "method of operation",[52] in that it is the natural means of navigation, and thus was not protected by copyright or patent. This concept of "method of operation" is the justification for several existing products and applications, such as automobiles, microwaves, and remote controls. These products have a natural usage pattern, thus restricting companies from owning patents for them. Companies can still patent particular innovations in the extension of their natural use, but cannot patent the overall product.

The court's reversal of the Lotus 1-2-3 vs. Borland Quattro case shifted the pendulum of software protection back in favor of competition by allowing companies to mimic existing applications and then continue to build on them. By using these tactics, it forced the original software company, whose products had been cloned, to respond to the competitor and offer more features, security, reliability, supportability, lower prices, etc. in turn to combat the newly released products. This is the environment that exists today and strongly favors the consumer, while at the same time allowing for new applications to be quickly adopted in the market. Since companies can legally clone existing applications, it allows for the release of products that simplify or extend these existing applications, working well in niche markets, such as specific vertical markets or particular user bases, such as educational software or "Mom & Pop shops". By easing the legal process of extending existing applications, the reversal of this lawsuit has created a more expansive software market. By establishing a distinction between the interface of an application and the

---

52 *Id. See also* Wikipedia "Lotus vs. Borland" - http://en.wikipedia.org/wiki/Lotus_v._Borland

implementation, as it pertains to copyright, the door has been opened to software developers to create clones of existing software and not infringe on any copyright laws.

There were, of course, some negative aspects to the decision. If a company had created an intuitive, intelligent user interface that the market responded well to, other companies could quickly mimic the interface and enter the market with minimal innovation. Similarly, as more companies create products with similar interfaces, it becomes harder for companies to release a new and different interface since it bucks the trend of the existing market. As a result, companies get lazy with their development in the interface arena and software maintains a similar look and feel for many releases, while software companies work on adding features as their product differentiation.

Other notable cases

Apple v. *Microsoft*[53] – In the late 1980's, Apple claimed a copyright on the look and feel of their Mac OS operating system and sued Microsoft for infringement with Windows. In the end, Apple lost the six-year lawsuit claiming that Microsoft's use of graphics within Windows infringed on Apple's Macintosh OS graphics usage. Even today, the new Vista operating system uses transparency in a similar technique as Mac OS X[54] as well as many other similarities.

Xerox v. *Apple*[55] – When Apple decided to sue Microsoft over stealing operating system ideas, Xerox decided to follow suit, claiming that the ideas Apple were claiming as their own actually were created by Xerox. Before creating the Mac OS, the designers at Apple were allowed to tour the Xerox facilities and explore the software that Xerox had created for their internal use. The Mac OS designers later claimed the Xerox software was influential in their design. Xerox took the opportunity to thus claim a piece of Apple's prize should they win the lawsuit against Microsoft.

Impact of Look & Feel Lawsuits

The three main look and feel lawsuits discussed caused mixed results in the innovation and competition of creating software. The initial Lotus/Borland ruling forced companies to drive innovation in new areas of user experience by restricting them from releasing software that mimicked existing applications. However, it also weakened some areas of applications where customers would be forced to buy products with necessary features even though the user interface and experience was much weaker than an existing product that lacked the feature. When the ruling was reversed, it quickly encouraged more competition since companies could enter into existing

---

[53] *Apple Computer, Inc. v. Microsoft Corp.,* 35 F.3d 1435 (9th Cir. 1994). See also: http://en.wikipedia.org/wiki/Apple_Computer,_Inc._v._Microsoft_Corp

[54] Wikipedia "Look and Feel" - http://en.wikipedia.org/wiki/Look_and_feel

[55] See Wikipedia "*Apple Computer, Inc. v. Microsoft Corp*", http://en.wikipedia.org/wiki/Apple_Computer,_Inc._v._Microsoft_Corp

markets more rapidly.

Java vs. .NET: An example of how not to exercise IP rights

In 1991, at Sun Microsystems, James Gosling created the "Oak" programming language which differed from commonly-used programming languages, resurrecting an old idea of targeting a "virtual machine". Normally the compilation of a programming language changes it from the high-level representation typed in by the programmer into an equivalent representation which consists only of terms exposed by the physical hardware (this second representation is called "assembly language" by a programmer or an "instruction set architecture" (ISA) by the hardware vendor.) The assembly language program representation will refer to physical characteristics of the hardware (such as named registers) and thus a program in assembly language is not portable to a computer which has a different ISA (because, as an example, every computer has differently-named registers.)

Ideally a programming language, such as C, is portable between ISAs because the programming language can be compiled into different assembly languages. Real programs, however, need to use features of the computer which are not encapsulated by the base programming language. The assembler (which translates from a programming language to assembly language) will handle the assignment of registers for the programmer but there are no concepts in the C language to directly handle, for example, communication with the user of the program—a concept known as I/O, which stands for "Input/Output".

Normally hardware abstractions such as I/O are handled by the computer's operating system (OS). The OS will supply what is called a "library" of functionality that is exposed to the programming language. While these libraries can theoretically be used by any programming language, there is usually a language which is a natural choice for an OS. Usually this is the C programming language (for Unix and Windows) but sometimes other languages are chosen (such as Pascal for the early Macintosh systems.) This library is known as a "runtime" and is written specifically for an OS. A program targeting the Windows C runtime will cannot be assembled into a program targeting the Unix OS without being "ported", i.e., rewriting non-portable parts of the program.

Value of the development experience

The value of a computer to its users is the programs that can be run on it. For a software company, the value of its OS is driven by the "development experience", that is, the quality of the programming languages and runtime libraries available for that OS (these are known in aggregate as an Applications Programming Interface, or an API.) For a hardware company such as Sun the value of the API as well as the OS is measured by their success in selling hardware. A good API motivates programmers into developing for a particular OS. Good applications drive sales of the OS and the hardware to run it on.

Oak targeted a virtual machine, normally referred to as a VM, which is a layer of abstraction isolating the program from the ISA. The advantage of a VM is that the programmer only needs to write the program once. Instead of compiling a program to target a particular ISA, it is instead compiled to target the VM. When the program is executed by the user, the program instructions are compiled again by a part of the VM known as a Just-In-Time compiler, or a JIT, producing native machine code. Likewise, the runtime libraries which expose the VM's API are common across architectures so that a program written for the VM will work for any conforming implementation of the VM. The advantage of writing a program once for many architectures was the main benefit Sun advertised for Oak.

In 1995 Sun rechristened Oak as the Java programming language and promised to provide Java virtual machines (JVMs) and API libraries for popular operating systems. The "Write Once, Run Anywhere" experience meant that Java would be a useful language for programmers to learn. The syntax of the language was derived from the massively successful yet overly complex C++ programming language. The similarity meant that Java was already familiar to a large number of programmers. The simplification meant that it was easy to learn. Java offered programmers an easy way to develop applications for a large number of operating systems and it offered OS and hardware vendors a large number of programmers who could develop applications.

Java's adoption driven by the Web

Java's adoption in the mass marketplace was guaranteed by its inclusion in Netscape's Navigator web browser which presided over the rise of the commercial internet. In the hype surrounding the net's rise, Sun claimed that the platform-independent nature of Java would make the web browser (as opposed to Microsoft Windows) the default applications platform. In order to succeed at deposing Windows, Sun needed Windows programmers to program against the Java API instead of the Windows API.

Microsoft at the time knew very little about the web. A book authored in 1995 by Microsoft's co-founder, Bill Gates, virtually ignored the internet in favor of Microsoft-owned networking technologies. But that changed quickly at the end of 1995 when Gates redirected Microsoft to become an internet-focused company.[56] In March 1996 Microsoft agreed to license Java from Sun for inclusion in their Internet Explorer web browser. Further, Microsoft decided to more tightly integrate the next version of Internet Explorer, version 4, into the Windows OS and to create a full Java development kit for creating Windows-based applications. Microsoft realized that it had made a mistake by discounting the rise of the internet. Accepting Java meant at least that Microsoft—and

[56] *Wikipedia: The Road Ahead,* http://en.wikipedia.org/wiki/The_Road_Ahead

Windows developers—would not be left behind.

The announcement in March of 1996 outlined Microsoft's plans for Java. They created a collection of technologies called "Jakarta" which included integration with MS Internet Explorer, an integrated development environment and integration with their Component Object Model (COM) technology. Microsoft was clear about their intentions to extend Java to interoperate closely with Windows. Brad Silverberg, a senior VP of Microsoft, said "Integrating the Java language with COM…brings a whole new dimension to Java: integration with existing applications, systems and technologies."[57] Microsoft's whole Java offering was tied up in a package called J++ which implied that it was an evolution of Java in the way that the C++ programming language was an evolution of the simpler C.

COM is a very important technology for Windows. It allows programmers in any language to communicate across processes to share data and objects. Word and Excel need to expose their functionalities as reusable components of software programs instead of as programs themselves. In doing so, other programs can make use of these components.

Sun's fight to standardize Java—or not

Sun announced at its JavaOne conference in 1996 that it intended to standardize Java.[58] Standardization is key to mass adoption, but can result in a loss of control. On the other hand, Sun felt that standardization offered no benefit to dominant companies because it opens up the market to other players. Sun had invented this technology on their own and had full rights to license the technology. If it became a standard, the playing field was leveled. If Microsoft—or any other company—made a better Java the market could quickly shift away from Sun.

Sun danced back and forth between standardizing Java and maintaining tight control. In 1997 they approached the European Computer Manufacturer's Association (ECMA), a standardization body which is responsible for a number of programming languages. They withdrew almost immediately, however, ostensibly because of changes to ECMA's Publicly Available Specifications (PAS) procedures which would limit Sun's ability to maintain control of a standardized Java.

Sun claimed that these changes came about because of lobbying by Microsoft and Hewlett-Packard (HP), two dominant members of the Windows-Intel ('Wintel') monopolies.[59] The idea of 'Wintel' is that the mass market of computers ran Windows on Intel hardware. Sun competes directly against both companies and sees this duopoly as a threat to their business model.

---

[57] *http://www.microsoft.com/presspass/press/1996/mar96/sunmspr.mspx*

[58] Egyedi, T.M., "Why Java was Not Standardized Twice" (hereinafter Egyedi) at 2 http://csdl2.computer.org/comp/proceedings/hicss/2001/0981/05/09815015.pdf

[59] Egyedi at 6

Microsoft's embrace

Microsoft actively extended the Java language and API in a manner which threatened the portability of Java. In October, 1997, Sun sued Microsoft for breach of contract. Sun specifically sought to enjoin Microsoft "from using the Java Compatible logo; from doing directly or indirectly any acts or making any statements that are likely to cause confusion, mistakes or deception in the marketplace as to the compatibility of IE (Internet Explorer) 4.0 and SDKJ (Software Development Kit for Java) 1.1 with Sun's Java technology . . . from doing directly or indirectly any acts that are likely to diminish the value of the Java Compatible logo. . ." They saw themselves as the "stewards of Java"—because Java had no standard to protect it—and felt that Microsoft was inducing software developers to write non-portable programs, while incorrectly believing that they would enjoy the benefits of "Write Once Run Anywhere" that Java promised. By extending the Java language Microsoft would eliminate the main benefit of Java to Sun: selling Sun hardware.

The technical issues were outlined as a failure to support the Java Native Method Interface (JNI) and Remote Method Invocation (RMI) interfaces as well as the nonstandard extensions to the Java API. COM integration was a substitute for JNI, though it was neither a complete nor compliant substitute. Because RMI focused on distributed computing while Microsoft focused on personal computing Microsoft had little interest in implementing RMI.[60]

Microsoft countersued, claiming that "Sun failed to deliver technology that passes Sun's own test suites and runs on the Microsoft Reference Implementation" and that Sun failed to treat Microsoft equally.[61] Interestingly, Sun's press release about the October 1997 lawsuit has a question and answer about the lawsuit. One of the questions is "Is Netscape any more compliant than Microsoft"? The answer Sun gives is that Netscape did not mislead people and has been open about its noncompliance. Microsoft was apparently being sued for hubris.

In November 1998 Sun created the Java Community Process and the Community Source initiative. This initiative opened the source of Java in that it was free of charge and accessible but decisions about the development of Java were ultimately bound to Sun's licensing agreements. [62] Sun was criticized by the Free Software movement for claiming to be open source (specifically, by Richard Stallman in his article "Free but Shackled—The Java Trap"[63]) but always maintained that the purpose of licensing was to protect Java developers. As James Gosling said, "The catch in the Sun Java source license is all about defending the needs of developers who work above the interface.

---

[60] *http://www.sun.com/smi/Press/sunflash/1997-10/sunflash.971007.10.xml*
[61] *http://www.microsoft.com/presspass/press/1997/oct97/msuncspr.mspx*
[62] *Id*. at 3.
[63] *http://www.gnu.org/philosophy/java-trap.html*

This ends up being constraining to folks who work under the interface, but in a way that is hugely beneficial to those who work above. We believe that for a developer who has built a Java application they have a right to trust that when some other developer says 'I have a Java VM for you to use', that their application will work."[64]

In April, 1999 Sun approached ECMA again to discuss standardization of Java. They asked ECMA to carry out "passive maintenance" of Java—essentially to certify Java as a standard but allow the Java Community Process (controlled by Sun) to control its development. ECMA refused but finally agreed to create a technical committee, TC41, which would standardize Java. Sun provided the main editor for the committee and emphasized that the focus should be on editing the language, not extending it. The committee was chaired by IBM and had a subcommittee on API chaired by Microsoft. In December, Sun withdrew and the committee decided not to go on without Sun.[65]

Microsoft's response

In November 1998 Microsoft responded to the preliminary injunction granted in Sun's copyright infringement lawsuit which required Microsoft to support the JNI and remove the keywords they had added to the Java language.[66]

Java was the solution Microsoft needed to allow programmers to develop efficient code (like C/C++ programmers) in a rapid-development style (like Visual Basic programmers). Java, and Microsoft's extensions to Java, allowed the Microsoft development platform to target the middle ground between their existing users. Microsoft's non-compatible J/Direct allowed Java programmers to access the Windows API in a way comparable to C/C++ and VB programmers—something not offered by JNI. The loss of the copyright lawsuit crippled Microsoft's ability to provide a competitive advantage to Windows for Java programmers because Sun's Java was not as technologically advanced.

In January 2001 Microsoft and Sun reached agreement, resolving Sun's October 1997 lawsuit. Microsoft agreed to pay Sun $20 million and to not use Sun's Java-compatible logo. In return, Microsoft could ship Java with its OS and all products related to Java for seven years.[67]

.NET Framework

In response to their inability to offer the Java development experience they wanted to offer Windows developers, Microsoft ceased development of the J++ product. To fill the space that Java

---

[64] *http://today.java.net/jag/page7.html#59*

[65] Egyedi at 7

[66] *http://www.microsoft.com/presspass/press/1998/nov98/sunpr.mspx*

[67] *http://www.microsoft.com/presspass/press/2001/jan01/01-23sunpr.mspx*

had occupied, Microsoft created an alternative to Java, the .NET Framework. Like Java, this framework is a VM-based platform which theoretically allows a WORA experience. Technologies invented by Microsoft like J/Direct and delegates (which are a kind of type-safe callback function, "one of the most useful programming mechanisms ever created", according to author Jeffrey Richter[68]) could be offered for programming on Windows without the restrictions and limitations imposed by Sun's licensing agreements.

The injunction of November 1998 asked for Microsoft's compliance but by 2002 it was apparent that Microsoft had responded with non-participation. This meant that Microsoft was distributing four year-old VM technology with Windows while the .NET Framework was brand-new. Users could get an alternative VM for Windows but it was well-established in the Microsoft antitrust case that it is onerous to expect users to download large programs as opposed to shipping them with the OS.

In January 2001 Microsoft shipped a version of their Visual Basic and C/C++ development platforms with support for the .NET Framework built-in as well as a new, very Java-like programming language called C#. All of these technologies were ECMA-standardized and thus open to modification without license agreements or copyright infringement. In March 2002 Sun moved for an injunction to force Microsoft to distribute Sun's version of the Java VM and to stop distributing Microsoft's outdated version. Sun claimed that Microsoft had realized that Java threatened the barriers to competition on the Windows platform and tried to seize control of the Java language. Forcing Microsoft to distribute Sun's Java runtime meant consumers and developers could decide on the merits of the technology.[69]

In April of 2004 Microsoft and Sun reached agreement on all pending disputes between the two companies, including disputes related to Java. Microsoft made a second punitive payment to Sun, this time of $1.6 billion. They agreed that Microsoft could continue to support Java and that the two companies would work together to "improve technical collaboration between their Java and .NET technologies."[70] But in these two years the landscape had changed significantly. In March 2002, Sun claimed that .NET was a response to Java, which Microsoft had "frantically struggled to develop." In the agreement of April 2004 Sun acknowledged .NET as a competitive alternative to Java. Sun had lost the advantage of having a main competitor relying on their copyrighted technology.

As a final note, in November 2006 Sun released a version of Java under the GNU General Public License version 2.[71] This is the second version of the license created by Sun's critic, Richard

---

[68] *http://msdn.microsoft.com/msdnmag/issues/01/04/net/*

[69] *http://www.sun.com/lawsuit/motion.pdf*

[70] *http://www.microsoft.com/presspass/press/2004/apr04/04-02sunagreementpr.mspx*

[71] *http://www.sun.com/2006-1113/feature/story.jsp*

Stallman, that James Gosling said constrained its users by limiting the protections Sun could provide them. This license removes Sun's control of the language much more than ECMA certification ever would have. Sun had won each battle but in the end they lost the war.

Conclusion

Software's interaction with the intellectual property regime in the United States has always been difficult. Open source has had mixed success. Trade secrets fall to reverse engineering in a clean room. Copyright does not provide clear protections for the look and feel of software. And licensing copyrighted pieces of a technology—as exhibited by Sun—can be a double-edged sword. It's fantastic to settle multiple lawsuits against your competitor. It's not so good to lose your market in the process.

But such interactions are not unusual for a relatively new field of endeavor. Some cases have fallen on the side of the loose-constructionists, those who favor strong IP protection for many aspects of software. Others have satisfied the narrow-constructionists, by defining areas of software which do not generate IP rights. Courts are still busily working out suits under the DMCA. Perhaps other forms of protection would be more suitable for software than copyrights have proven to be, but Congressional history with legislation does not leave one hopeful that such solutions will be adopted. At this point in time, an uneasy balance exists in the world of software. Copyright suffices because it must.