SOFTWARE PATENTS: PROBLEMS AND SOLUTIONS

David Stein, April 2001

I. INTRODUCTION

The patent system has existed throughout our nation's history, stemming from the Constitution[1], which was founded on historical patent systems developed in England and ancient Venice.[2] However, the purpose of patents was not to protect scientific development in general, but rather to promote practical discoveries that might benefit the common man. Therefore, Congress enumerated specific classes of patentable inventions in the Patent Act of 1836 geared toward *useful* inventions, such as machines, manufactures, and compositions of matter, and the federal courts and the U.S. Patent and Trademark Office (hereinafter "patent office") uniformly rejected patents for "mental processes" or other scientific principles, absent some attachment to a tangible, useful invention.

The changing face of technology has recently presented several quandaries for the patent system. Cutting-edge industries are producing "inventions" that do not fit within the statutory classes, but it appears that they *should* be patentable, because they have taken on the characteristics of more traditional inventions. One example is biotechnology; such inventions as DNA sequences, in the form of novel genes and proteins, and new forms of life, both microscopic and macroscopic, have proven so useful to science and industry that the patent office has surrendered its prior rejection of these

[1] *See*, U.S. CONSTITUTION Art. I § 8 (Congress is granted the reserve power "To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries….")
[2] *See generally*, BRUCE W. BUGBEE, THE GENESIS OF AMERICAN PATENT AND COPYRIGHT LAW 8 (1967).

"inventions." Another example is software, which has rapidly evolved from its primitive roots (not much different from operating a handheld calculator) into an extremely varied, useful, and profitable industry.

The patent office faced a dilemma with software inventions. On the one hand, the complexity and utility of software for virtually all industries suggested that software should be patentable. On the other hand, software inventions consist of a set of steps, much like mental processes, and on first glance do not appear to fit the statutory classes set forth by the patent act. Furthermore, the nature of the software industry complicated a determination of the patentability of any particular software invention claimed in a patent application. For these practical reasons, the patent office maintained a categorical rejection of software inventions for failure to fit a statutory class during the early years of the software industry.

The federal courts were not so discouraged by the difficult reality of software patent examination. Over the course of the late 1980's and early 1990's, the federal courts issued a series of opinions that first lowered, and then eliminated, the categorical barriers that the patent office had erected against software patents, issuing a firm declaration that software inventions were just as patentable as any other useful invention and implicitly ordering the patent office to evaluate them accordingly.

This reversal caught the patent office by surprise. Its examiners were unskilled in the field of software; its prior art library was virtually empty of software inventions; and its organization was not conducive to carving out a whole new area for the examination of software inventions. Thus, it was unprepared for the avalanche of software patent applications that followed, and its examiners could conduct only the most cursory and

basic examination. The result was a glut of software patents that have issued, and continue to issue, but are questionably valid.

This outpouring of meritless software patents has had a catastrophic effect on the software industry. Development is hampered by the fear of infringing a swath of software patents by using well-known and oft-employed software techniques, and are unable to combat invalid patents because of the prohibitive costs and uncertainty of litigation. The flood of software patents is acting as a barrier to entry, an ominous threat to small and independent developers, and a grave danger to the open-source movement. And, the industry opinion of software patents has eroded to the point where critics call software patents a "registration system," whereby patents are granted merely upon filing an application.

The purpose of this paper is to explore the problems facing software patents. The events that led to the current system and the problems with obtaining, enforcing, avoiding, and defending against software patents will be presented. Finally, potential solutions, ranging from radical changes (such as abandoning the software patent system entirely) to legislative changes (such as changing the term of a software patent) to tactical changes (such as a cooperative industry effort to police software patents), will be explored and critiqued.

II. THE STATUS QUO: EARLY REJECTION OF SOFTWARE PATENTS

Patents are traditionally granted for useful inventions, but the invention must meet certain broad criteria set forth in the patent laws. In particular, 35 USC §101 sets forth

the statutory classes for patentable subject matter as a "machine, manufacture, composition, or process, or a combination of these inventions"; inventions that do not fall within these classes are implicitly excluded from the realm of patentable subject matter.

One category of invention that the patent office traditionally rejected as falling outside these classes was a "mental process" – a cognitive method for reaching a useful result.[3] Even if the mental process could be embodied in a machine that carried out the same steps to reach the same result, and even if the patent office would have granted a patent on this machine, nevertheless the functionally identical mental process was categorically banned from patent protection. This discrepancy had been consistently applied throughout the history of the American patent system. One of the earliest Supreme Court patent decisions held that "a patent may be for a new and useful art; but it must be practical, it must be practicable and referable to something by which it may be proved to be useful; a mere abstract principle cannot be appropriated by patent."[4] Not long before the first software cases appeared before the courts, the Court of Customs and Patent Appeals (hereinafter "CCPA"), the precursor to the Federal Circuit, affirmed this principle, finding it "self-evident that thought is not patentable."[5] On the other hand, the

[3] Although one of the statutory classes is a "process," this term was traditionally limited to a method of making or using a machine, manufacture, or composition of matter – in short, "processes" were only patentable if attached to an independently patentable invention. For example, the patent office has uniformly rejected applications that claim recipes as "processes" for making food. In short, the statutory class of "process" was very limited, and did not include purely "mental" processes, such as the steps for long division. This distinction is important because, although software seems very similar to a "process," the patent office traditionally denied patentability because software seems like a "*mental* process," as will be discussed *infra*.

It should further be noted that the traditional limitation of the "process" statutory class has been significantly weakened in recent years. Patents are now being issued for "processes" traditionally rejected, such as a "method of putting" in golf (U.S. Patent No. 5,616,089). However, this general shift occurred simultaneously with or subsequent to the relaxation of the restrictions on software patents, so software patent applicants could not take advantage of this change in arguing the patentability of software inventions.

[4] Evans v. Eaton, 16 U.S. 454, 475 (1818).

[5] In re Abrams, 188 F.2d 165, 168 (CCPA 1951).

patentability of machines hearkens back to the very first patent act, which explicitly includes machines in the realm of patentable inventions,[6] and this class has been consistently maintained through every revision of the patent laws and into its current version.[7]

At first glance, it may seem paradoxical to allow a patent for a machine carrying out a particular process and categorically reject a patent for those same steps carried out by a human, but the reasons for this dichotomy are practical and sound. First, compared with the intangible workings of the human brain, a machine presents a much better fit with the traditional concept of an "invention." The end product of a machine invention can be demonstrated for, licensed to, or sold to others, who can then use it without an understanding of the fundamental operation of the device. Therefore, a machine is often a black box, a tool for carrying out a specified purpose without concerning its operator with the specifics of the processes being used. Additionally, when the owner of a machine conveys it away, he must build or acquire another machine to make use of the invention again; similarly, purchasers or licensees of a patented machine cannot pass along the invention to third parties without simultaneously surrendering the ability to use it. On the other hand, a mental process cannot be demonstrated and conveyed like a device, and using the process necessarily requires an understanding of the principle. It can be replicated at will without the material costs normally associated with manufacturing a device, and sharing the knowledge does not deprive the teacher of the

[6] Patent Act of 1790 § 1 ("[U]pon the petition of any person or persons to the Secretary of State, … setting forth, that he, she, or they, hath or have invented or discovered any useful art, manufacture, engine, *machine*, or device, or any improvement therein not before known or used, and praying that a patent may be granted therefor, … letters patent [may] be made out in the name of the United States…." (emphasis added.))

[7] 35 USC § 101 ("Whoever invents or discovers any new and useful process, *machine*, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefore, subject to the conditions and requirements of this title." (emphasis added.))

ability to use it. Thus, patentees cannot reliably prevent licensees from passing the information to third parties or releasing the information to the public, and can easily be deprived the financial benefits of the patented invention. Therefore, a mental process therefore lacks many of the tangible properties and practical considerations that are inherent in the types of "invention" for which patents are granted.

Furthermore, the public nature of patents makes enforcement of "mental process" patents more problematic than for machine patents. In the case of a machine, practicing the claimed invention requires more than merely understanding the invention; the design, construction, and testing of the machine constitute a nontrivial part of the design process. But for mental processes, an understanding of the principle is all that is needed to carry out the invention; no development or manufacturing costs are necessary. This difference complicates enforcement of patent rights, because the principle of the invention is publicly disclosed when a patent is published. Therefore, a competitor could infringe a "mental process" patent simply by reading the specification and carrying out the disclosed steps. And, because a "mental process" invention is, naturally, as fleeting and intangible as thought itself, evidence presented by a patentee in an infringement suit must consist of circumstantial evidence and conjecture; such infringement cases are understandably difficult to prove and adjudicate. On the other hand, a competitor seeking to infringe a machine patent must design, manufacture, and test the machine; not only do such steps serve as an additional barrier to infringement, but the end product is damning, tangible evidence of infringement that the patentee may later use in infringement proceedings. Even detecting an infringement of a mental process is necessarily more difficult than for machines.

For these reasons, patentability for inventions depends, at least in part, on the tangibility of the invention: physical creations like machines find more favor with the patent office than intangible concepts, such as mental processes. The Supreme Court affirmed this critical characteristic: "While a scientific truth, or the mathematical expression of it, is not patentable invention, a novel and useful structure created with the aid of knowledge of scientific truth may be."[8] Therefore, abstraction can be seen as a spectrum, with patentable, concrete inventions at one end and unpatentable, abstract inventions at the opposite end.

In this spectrum of abstraction, software occupies a difficult middle ground. A machine containing a general-purpose processor that runs software is certainly a tangible creation, and clearly falls within the patentable categories of inventions set forth by 35 USC § 101. But the software itself is wholly intangible, existing as a set of data embedded inside the machine on some form of medium (a hard disk drive, a floppy disk or CD-ROM, or a memory chip.) The only evidence of its existence is that a computer containing the software behaves in a particular way, whereas a computer that lacks the software does not. Even a printed copy of the computer code of a program is, at best, an abstract precursor of the real invention, and the PTO only grants patents on inventions that have been successfully reduced to practice from a precursor to a physical, working invention.[9]

[8] Mackay Co. v. Radio Corp., 306 U.S. 86, 94, 59 S.Ct. 427, 431, 83 L.Ed. 506 (1939).

[9] For example, the patent office will not grant a patent for an amino acid sequence or a machine blueprint; even though these inventions could successfully be used by any person "skilled in the art" to create a useful protein or machine. The inventor must demonstrate that the disclosure is enabling by successfully using it to create the claimed invention, and offering proof of this use during patent prosecution. At most, the amino acid sequence or machine blueprint may be used in the patent as an abstract representation of the physical invention for which the patent has been granted. Similarly, a printout of a computer program can only be used to create this pattern of data that results in a particular behavior in a general-purpose machine, and is therefore only a "blueprint" of the actual program.

A real-world example of this spectrum of abstraction is illustrated by a comparison of different methods of carrying out long division. An electric circuit, a software program, and the human mind can all carry out long division, but each does so in a different manner. A circuit conducts electricity through preset, fixed logic gates arranged specifically and permanently for this purpose. The human mind has no permanent structures for mathematical operations; it must first access the memory part of the brain that remember how to carry out long division, and then cause another part of the brain to follow these steps for the purpose of carrying out long division. Software represents an amalgam of these methods: the memory part of the machine holds the steps required to perform long division, and the hardware part of the machine follows these in a specific order to calculate the dividend. Rather than passing electricity through preset structures, the machine must *indirectly* carry out the steps by concurrently performing each step specified by the software; the result is, in one sense, a by-product of the main task of the computer in performing these steps. Thus, on the one hand, the software is manipulating circuits and electricity in much the same way as a hard-wired electric circuit; on the other hand, the processor cannot carry out the operation without indirect reference to the stored instructions, and the claimed invention lies in the abstract, recipe-like set of ordered steps that the general-purpose computer performs.

When pressed to grant patents for software inventions, the patent office and the federal courts were required to decide whether, on this spectrum of abstraction, to group software with mental processes and reject patent applications categorically, or to group software with machines and allow patents. The early decisions of the federal courts seized on the indirection – the fact that the claimed invention only constituted a set of

stored instructions, to which a machine referenced in order to perform indirectly the claimed invention – as grounds for grouping software with mental processes and rejecting patentability.

The most frequently cited origin of the early prohibition of software patents is the Supreme Court decision in *Gottschalk v. Benson*.[10] The Court was presented an appeal by an inventor for a method of converting between two common mathematical representations of a number. Computer memory is digital and divided into bytes, each of which can be used to represent 256 different numbers. While using all of the bits in a byte to store a number in base-two format ("binary" format) makes the best use of memory, such numbers are not easily readable by humans, who are more accustomed to base-ten representation. Alternately, numbers could be stored by using each byte to represent only one base-ten number ("binary-coded decimal" format, also known as BCD), but this is much less efficient and large numbers require much more memory to store in this format than in base-two format – a critical concern in an era when computer memory was very expensive. A better solution to this tradeoff between readability and efficient storage was to store all numbers in binary format and use a method to covert the numbers to BCD when a user wanted to read the number, and for converting BCD input by a user into a binary representation that the computer could store efficiently. Such a method was designed by the appellant and claimed in a patent application, containing the following claims:

> 8. The method of converting signals from binary coded decimal form into binary which comprises the steps of
> (1) storing the binary coded decimal signals in a reentrant shift register,
> (2) shifting the signals to the right by at least three places, until there is a binary "1" in the second position of said register,

[10] 409 U.S. 63 (1972).

(3) masking out said binary "1" in said second position of said register,
(4) adding a binary "1" to the first position of said register,
(5) shifting the signals to the left by two positions,
(6) adding a "1" to said first position, and
(7) shifting the signals to the right by at least three positions in preparation for a succeeding binary "1" in the second position of said register.

13. A data processing method for converting binary coded decimal number representations into binary number representations comprising the steps of
(1) testing each binary digit position "1," beginning with the least significant binary digit position, of the most significant decimal digit representation for a binary "0" or a binary "1";
(2) if a binary "0" is detected, repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;
(3) if a binary "1" is detected, adding a binary "1" at the (I+1)th and (I+3)th least significant binary digit positions of the next lesser significant decimal digit representation, and repeating step (1) for the next least significant binary digit position of said most significant decimal digit representation;
(4) upon exhausting the binary digit positions of said most significant decimal digit representation, repeating steps (1) through (3) for the next lesser significant decimal digit representation as modified by the previous execution of steps (1) through (3); and
(5) repeating steps (1) through (4) until the second least significant digit representation has been so processed.

The patent office rejected this application, but the CCPA reversed and held the invention unpatentable.

The Supreme Court reversed the decision of the CCPA and rejected the claims. The Court first noted the longstanding categorical rejection of applications for patent on mental processes. The Court then rejected the claims for overbreadth:

> Here the 'process' claim is so abstract and sweeping as to cover both known and unknown uses of the BCD to pure binary conversion. The end use may (1) vary from the operation of a train to verification of drivers' licenses to researching the law books for precedents and (2) be performed through any existing machinery or future-devised machinery or without any apparatus.[11]

The applicant argued to the Supreme Court that a rejection of his application would act as a categorical ban on all software programs. The Court denied the breadth of this

[11] *Id.* at 68.

conclusion, but then stated that it would not allow a patent on a "mathematical formula [that] has no substantial practical application except in connection with a digital computer," because this would "wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself."[12] And despite having explicitly reserved the question whether *any* computer program could be patentable, the Court deferred this decision to Congress and cited with approval the Commission on the Patent System, which advised a categorical ban on software patents.[13]

The consensus among the patent office and patent practitioners as a result of *Gottschalk* was that software was considered a mere "algorithm," or the digital equivalent of a mathematical formula or mental process that fell outside the statutory subject matter for which patents were granted. As a result, software developers turned to alternative methods of protection to protect their programs from copying. The protections traditionally employed, often concurrently and still in heavy use today, are trade secret and copyright. Because the consequences of this decision bear critically on later problems with software patents, a brief examination of these forms of protection is helpful.

Developers first sought to maintain the secrecy of their inventions by withholding the computer program and distributing only the compiled machine code to the public.[14]

---

[12] *Id.* at 71.

[13] *See*, "To Promote the Progress of … Useful Arts," REPORT OF THE PRESIDENT'S COMMISSION ON THE PATENT SYSTEM (1966), cited in *Gottschalk*, 409 U.S. at 72 n. 4 (1996) ("Uncertainty now exists as to whether the [patent] statute permits a valid patent to be granted on patents. … The Patent Office now cannot examine applications for programs because of a lack of a classification technique and the requisite search files. Even if these were available, reliable searches would not be feasible or economic because of the tremendous volume of prior art being generated. Without this search, the patenting of programs would be tantamount to mere registration and the presumption of validity would be all but nonexistent.")

[14] *See*, U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE AND INTELLECTUAL PROPERTY – BACKGROUND PAPER 23 (Global Press Publications, 1990) ("Trade secret has been the traditional favorite form of protection for mainframe and minicomputer software. From the viewpoint of a

Machine code is not only sufficient for the customer to use the invention, but is also difficult for competitors to analyze and reverse-engineer, especially with complicated programs. Machine code breaks down each complicated instruction in the computer program into many simple steps that the computer can perform, but reconstructing the original instruction from the steps can be time-consuming and very difficult. Moreover, documentation in a computer program that may help software developers study and understand the underlying software invention is discarded when the program is "compiled" into machine code, and therefore is withheld from competitors. For example, data in a computer program may be referred to by a variable name that describes what the data represents and how it is to be used, but the names are replaced with generic memory addresses in the machine code, and comments in the program that describe the purpose and consequences of performing particular instructions are not incorporated in the machine code. For these reasons, a software developer can provide strong protection of a software invention by keeping the program secret and only releasing the machine code to the public.

Software developers have also relied on the protection afforded by copyright. Because computer programs are developed by typing instructions into a computer, often using a program that is very similar to a word processing program, a computer program is a "writing" for the purpose of copyright law. As a result, a software developer is granted the same rights over his computer program as authors hold over their written works. When a software developer sells a copyrighted program to a user, the user may only use

---

software developer, the advantages to trade secret are that it protects a program's underlying ideas, logic, and structure, not just expression (as in copyright). It avoids formalities of registration or application and lengthy waits for protection. Enforcement is relatively clear-cut, and injunctions or compensatory relief is available for those who can prove misappropriation of trade secrets.")

the software in ways set forth by the license agreement (although the courts have granted some rights to licensees, such as reverse-engineering and the doctrine of "fair use," that cannot be restricted by the license agreement.) The advantages of copyright protection include its immediate effectiveness, lack of expense, and the ability to obtain copyright protection in conjunction with either a software patent or trade secret protection. However, copyright does not protect the underlying workings of a program; a competitor who independently develops the same invention or reverse-engineers the copyrighted product is free to use the invention in its own products unless some further form of protection exists.[15]

Thus, the early decisions by the patent office and the courts acted as a categorical bar to patents on software inventions. This would change unexpectedly over the next decade and bring with it an unforeseen assortment of problems.

III. THE CAUSE: SUDDEN ALLOWANCE OF SOFTWARE PATENTS BY THE COURTS

A. MODERATE RELAXATION OF PROHIBITION: *DIAMOND V. DIEHR* AND THE FREEMAN-ABLE-WATER TEST

The early rejection of applications for patents on software inventions rested, in large measure, on the nature of software development and use. In the early stages of

[15] *See,* Victoria A. Cundiff, "Protecting Computer Software as a Trade Secret," *reproduced in* PETER BROWN, WAYNE E. WEBB, JR. 18TH ANNUAL INSTITUTE ON COMPUTER LAW 2 (Practicing Law Institute 1998) ("Generally speaking, the copyright law protects literal expression, but not the ideas underlying that expression. Thus, if two software developers working independently each decided that there would be a market for software to generate computer graphics that transform two dimensional images into 3-D images, both could write code to implement that idea. While the specific code and the resulting output might both be copyrightable, neither developer could prevent the other from simply marketing any software to make 3-D images or even, most likely, from writing software to implement that idea using a particular order of activities (e.g., first expand the image, then rotate it, then construct linking lines between the original and rotated images.)")

software development, a software programmer would create an invention by providing instructions to a computer to be carried out sequentially and produce the desired result. This process was very similar to using a calculator or adding machine to run through a set of mathematical operations and compute a useful result; while it is clear that the adding machine or calculator clearly constituted a patentable machine, the method of *using* the device for any other purpose was not patentable, because the device would only serve as a tool for carrying out an unpatentable mathematical formula to compute an unpatentable mathematical result. The only observable differences were that with a calculator, the steps are executed immediately upon entering them, but in a computer the steps are stored in memory and then executed at the behest of the user; and that a computer program might be entered in a more "natural" and readable language, such as BASIC, and more complex operations were possible. Nevertheless, the heart of software consisted of a set of sequential mathematical operations not fundamentally different from the operation of a calculator, and thus remained unpatentable.

However, as the capabilities of general-purpose computers increased dramatically, the nature of software began to change as well. Software escaped its abstract, theoretical shackles and began to have real-world impact in several ways. Software was developed to monitor and control real-world devices in manufacturing. Computers controlled real-world sensors and performed useful, real-world functions on their measurements. No longer could general-purpose computers be seen merely as the more powerful, but nevertheless closely related, kindred of calculators and adding machines; rather, software began to provide functions that were at first practically useful, and then indispensable. While the roots of this trend may have been disputable in the 1980s, today few can deny

that general-purpose computers and software play an integral and irreplaceable role in every facet of our economy and businesses.

Furthermore, as the breadth and robustness of software expanded, the process of developing software became similar to the process of developing patentable inventions. Early software inventions could be created simply and quickly, and perhaps the initial reluctance of the patent office and the courts to allow software patents was due, in part, to the simplicity of software development and the limited scope of tasks that software could perform. These early computers required a programmer to write programs that directly controlled the hardware. Not only did this require the programmer to learn the intimate details of how to control the hardware, but any simple task had to be performed by specifying every single step the hardware must perform. This took a great deal of time, and the limited memory of early computers made complex tasks consisting of millions of steps unfeasible.

Today's software development is a very different process. The programming code of a typical application does not communicate directly with the software. Rather, the programmer passes instructions to the "software libraries" provided with a "compiler"; the compiler translates those references to its library functions into messages to the operating system; and the operating system converts those messages into generic instructions and communicates them to "device drivers" that directly control the hardware. This scheme allows the end developer to write a very simple, easy-to-understand statement that performs very complex operations, often consisting of thousands of machine-level instructions, on a wide range of very dissimilar operating

systems and computer hardware.[16] Furthermore, this complexity has created the "middleware" software industry, where developers produce tools, such as compilers, hardware drivers, and advanced programming interfaces, for other developers to use in creating software; and these competing "middleware" standards often necessitate the development of software standards, such as TCP/IP, that allow end-level developers to communicate with mid-level and bottom-level hardware in predictable ways.

Such multi-layered development processes, industry standards, and mid-level industries are, of course, commonly present in conventional manufacturing industries. End-level manufacturers in traditional industries employ a complicated design process and rely on a well-established set of standards to use hardware purchased from one manufacturer (similar to a software compiler) on materials purchased from another manufacturer in order to create an end product, and each of these tools and materials may be produced from other, more fundamental parts (similar to device drivers) and rely on other sets of standards. This multi-layered complexity in both industrial manufacturing and software development requires highly skilled engineers, very complex and intricate product planning and development, and high development costs; consequently, business practices traditionally used in manufacturing are now applicable and beneficial for software.

Finally, perhaps the strongest argument in favor of software patents is that the everyday utility of some software inventions has risen to equal, or even exceed, the utility of more conventional inventions; consequently, many useful software inventions *deserve*

[16] In fact, the development process is usually even more complicated and multi-layered than this. Some programming interfaces, like Java, allow high-level software to be executed on a wide variety of operating systems; others, like Microsoft's DirectX, allow software to perform very powerful functions directly on a wide range of peripheral hardware.

to be treated like patentable devices. A classic example is the patent for RSA encryption; although it is, in essence, a mathematical operation carried out in software, the practical utility of the invention is unquestionable. Before the development of the RSA algorithm,[17] security experts faced a common problem in encoded messages: Once you pass an encrypted message, how can you securely instruct the recipient on how to decode the message? Were the instructions for decoding the message to fall into the hands of an adversary, the point of encoding the message would be lost. Furthermore, encoding the instructions then led to the recursive problem of how one can securely teach the recipient how to decode the instructions for decoding the original message, and so on. At some point, the sender must transmit an uncoded message about how to decode something, and if that uncoded transmission were intercepted, the whole encoding scheme would be compromised.

The RSA algorithm presents an elegant solution to this problem. The algorithm provides two numeric "keys": a "public" key that is shared freely with the public and a "private" key that is kept secret by the sender. Any message encoded with one's public key can only be decoded by using that same person's private key, and any message encoded with the private key can only be decoded by using the same person's public key. This allows the following three scenarios:

1) A sender can encode a message with the recipient's *public* key before sending it. Then, no one – not even the sender – can decode the message without using the recipient's private key. In this case, an encoded message can be delivered secure in the knowledge that *only* the recipient can open and read the message.

[17] The algorithm is named after its inventors: Ronald Rivest, Adi Shamir, and Leonard Adelman.

2) A sender can encode a message with his own *private* key before sending it. Then, anyone can decode the message using the sender's *public* key – but everyone can be assured that *only* the sender could have encoded it, because only the sender knows the private key that must have been used to encode this message. Therefore, the message is openly readable to all, but is "digitally signed" by the sender.
3) A sender can encode a message first with his own *private* key, and then encode it with the recipient's *public* key, before sending it. When the recipient receives it, he decodes it with his own *private* key, and then with the sender's *public* key. This combines both security features of the algorithm: only the recipient can read the message, and only the sender could have written the message. This provides for maximal security.

Not only is this encryption scheme very strong,[18] it has become the *de facto* standard for encryption on the Internet and is heavily used to secure electronic transactions between Internet-based businesses and customers; indeed, it may be credited with having helped spark the development of the Internet business model. This sort of software invention is, admittedly, comprised essentially of an algorithm; but its practical, everyday value cannot be denied, and indeed, this algorithm appears to be exactly the sort of useful,

[18] The encryption uses a particularly difficult mathematical algorithm. One can run the uncoded message through a very simple algorithm, using a particular key, in order to derive the coded message that can only be decoded with the complementary key. But the algorithm is constructed in such a way that reversing this process – using a coded message and the original key to determine the original message – is *extremely* difficult, requiring immense computing power. Furthermore, reversing the algorithm gets exponentially more difficult when longer public and private keys are used. Although the algorithm was developed in 1977, it remained virtually unbreakable until 1999, when a few methods of cracking the encryption more quickly proved successful (one developed by Adi Shamir, one of the original inventors of RSA.) Nevertheless, even the simplified process requires extreme computational power; therefore, the algorithm is still considered strong enough for everyday use, and more difficult variations on the RSA method remain unbreakable.

valuable innovation that the patent laws are intended to protect.[19] Indeed, the patent office recognized its utility and granted a patent on the algorithm, which remained essentially unbroken, and thus valuable, until the end of the patent term.[20]

These changes in the software development process and in the real-world nature and impact of software led the Supreme Court to soften its former stance against software patents as articulated in *Gottschalk*. In *Diamond v. Diehr*,[21] the Supreme Court was faced with a new method for curing synthetic rubber by heating the liquid synthetic in a mold at a precise temperature for a precise amount of time; a properly cured product retains its shape and function under a wide range of conditions. The temperature and duration of the heating were determined using the Arrhenius equation, a well-known mathematical formula discovered in the 1880's[22] and widely used in the industry. Common industry practice dictated computing the results of this equation by hand and controlling the machinery manually, but this led to imprecise measurements, inexact operation of the machinery, and rubber products that were not optimally cured.

The new method claimed by Diehr in his application for patent was the use of software on a general-purpose computer to compute the Arrhenius equation and to control directly the manufacturing machinery. By using this equipment, Diehr controlled the curing process with greater precision and was able to achieve optimal curing consistently. This represented a considerable improvement in the rubber industry and a

[19] *See*, Bruce Parens, "Bruce Parens on Patents and Politics," *available at* http://sendmail.net?feed=interviewperenspatents ("That leaves the patents that actually do represent invention. RSA is the example here: the RSA guys invented public key cryptography, figured out the math to do it, successfully patented it – did real work, did real research.")
[20] U.S. Patent No. 4,405,829 was granted to RSA Security, Inc. for the algorithm on September 20, 1983 during a period where patent terms lasted for 17 years; although the patent was due to expire on September 20, 2000, RSA Security surrendered the patent to the public domain two weeks early as a show of goodwill.
[21] 450 U.S. 175 (1981).
[22] Svante Arrhenius, "On the Dissociation of Substances Dissolved in Water," *Zeitschrift fur physikalische Chemie* 631 (1887).

specific, efficient solution to a previously unsolvable problem. Nevertheless, the patent examiner decided that the claimed invention was simply a software implementation of the Arrhenius equation constituting nonstatutory subject matter and rejected the application as not fitting the statutory classes set forth in 35 USC § 101, and the Patent and Trademark Office Board of Appeals, the precursor to the current Board of Patent Appeals and Interferences (hereinafter "Board"), affirmed the rejection.

The CCPA reversed the rejection, and on appeal by the Commissioner, the Supreme Court affirmed patentability. The Court first held that regardless of the fact that a general-purpose computer and software involved, Diehr's invention fell within the statutory class of a "process" because of its similarity to traditional industrial manufacturing processes. The Court distinguished its former opinions on software, including *Gottschalk*, by stating that those holdings only reaffirmed the principle that ideas and mathematical formulae, in the absence of physical inputs or outputs, are unpatentable. In contrast, the Court held that Diehr's claim was patentable because it was connected to a tangible result:

> [T]he respondents here do not seek to patent a mathematical formula. Instead, they seek patent protection fro a process of curing synthetic rubber. Their process admittedly employs a well-known mathematical equation, but they do not seek to pre-empt the use of that equation. Rather, they seek only to foreclose from others the use of that equation in conjunction with all of the other steps in their claimed process. … Obviously, one does not need a "computer" to cure natural or synthetic rubber, but if the computer use incorporated in the process patent significantly lessens the possibility of "overcuring" or "undercuring," the process as a whole does not thereby become unpatentable subject matter.[23]

The Court rejected the contention that an invention becomes unpatentable solely because part of the process utilizes a well-known mathematical formula or calls for the use of a

[23] *Id.*, at 187.

computer or software. Rather, the invention may be patentable "when a claim containing a mathematical formula implements or applies that formula in a structure or process which, *when considered as a whole*, is performing a function which the patent laws were designed to protect…."[24]

In so holding, the Court affirmed the patentability of using a general-purpose computer and software in an industrial process. Subsequent court decisions focused on the Supreme Court's basis for differentiating "purely mathematical" software from software that had some real-world impact. These decisions formulated a test, known as the Freeman-Walter-Abele test,[25] which stated that computer software was patentable if one of the two conditions were satisfied: where the software operates on physical measurements, or where the software produced real-world, "post-solution activity."[26] This test ensured that the mathematical operations carried out by patentable software had some real-world connection and was not merely a mathematical manipulation of data.

B. COMPLETE RELAXATION OF PROHIBITION: *IN RE ALAPPAT* AND *STATE STREET BANK*

The Freeman-Walter-Abele test was short-lived. The courts soon abandoned the test in favor of a more traditional standard that required the patentability of the invention to be "considered as a whole" without focusing specifically on the real-world input or output.

[24] *Id.*, at 192 (emphasis added.)
[25] *See* Application of Freeman, 573 F.2d 1237 (CCPA 1978); In re Walter, 618 F.2d 758 (CCPA 1980); In re Abele, 684 F.2d 902 (CCPA 1982).
[26] Freeman, 573 F.2d at 1246.

The first decision to do so was the holding of the Court of Appeals for the Federal Circuit (hereinafter "Federal Circuit") in *In re Alappat*.[27] The court was faced with an appeal of the rejection of a patent application claiming a method of improving the displayed results of a digital oscilloscope. Before Alappat's invention, drawing a diagonal line on a digital display, such as an oscilloscope, was difficult because the screen displayed a grid of points ("pixels"); diagonals had to be drawn with a stair-step approach, leading to a jagged appearance. Alappat's invention also colored the pixels adjacent to the line with a dimmer color, producing a smoother appearance to the line. The process, commonly known as "anti-aliasing" and heavily employed in modern graphics applications to smooth the appearance of borders in three-dimensional simulations, was described by Alappat in his patent claim:

> A rasterizer for converting vector list data representing sample magnitudes of an input waveform into anti-aliased pixel illumination intensity data to be displayed on a display means comprising:
>
> (b) means for determining the vertical distance between the endpoints of each of the vectors in the data list;
> (c) means for determining the elevation of a row of pixels that is spanned by the vector;
> (d) means for normalizing the vertical distance and elevation; and
> (e) means for outputting illumination intensity data as a predetermined function of the normalized vertical distance and elevation.[28]

In short, Alappat's software invention operated on data representing a line and provided a method for displaying the data as a line that looked much more natural and smoother than other methods of displaying the line. Therefore, it lacked both "pre-processing input" and "post-processing activity" that was sufficiently attached to the real world to pass the Freeman-Walter-Abele test. On this basis, the examiner held that the invention essentially constituted a "mathematical algorithm" and rejected the application; an en

[27] 33 F.3d 1526 (CAFC 1994).
[28] *Id.*, at 1558-39.

banc panel of the Board initially reversed and allowed the application, but on reconsideration, an expanded panel of the Board affirmed the rejection.

The Federal Circuit reversed and affirmed patentability. The court first reiterated the limitation of patentable subject matter to the statutory classes and the unpatentability of "abstract ideas." The court then opted to analyze the patentability of Alappat's invention by considering the invention as a whole, without even mentioning the criteria set forth in the Freeman-Walter-Abele test: "the dispositive inquiry is whether the claim *as a whole* is directed to statutory subject matter…."[29] The Federal Circuit thus formulated the test for patentability:

> [T]he proper inquiry in dealing with the so-called mathematical subject matter exception to § 101 alleged herein is to see whether the claimed subject matter as a whole is a disembodied mathematical concept, whether categorized as a mathematical formula, mathematical equation, mathematical algorithm, or the like, which in essence represents nothing more than a "law of nature," "natural phenomenon," or "abstract idea." If so, Diehr precludes the patenting of the subject matter.[30]

Under this analysis, the court held that Alappat's application stated a patentable invention that produced a "useful, concrete, and tangible result."[31] Although the reference to a "tangible result" could be read as a requirement that the claimed invention have some physical, real-world consequence (as set forth in the Freeman-Walter-Abele test), not only does this invention clearly lack any such input or output beyond the display output of an oscilloscope, but the court specifically states that "[t]he fact that the four claimed elements function to *transform one set of data to another* does not alone justify a holding that the claim as a whole is directed to nonstatutory subject matter."[32]

---

[29] *Id.*, at 1543, *citing* Diehr, 450 U.S. at 187.
[30] *Id.*, at 1544.
[31] *Id.*, at 1544.
[32] *Id.*, at 1544, *citing* In re Iwahashi, 888 F.2d 1370, 1375 (CAFC 1989).

In so holding, the Federal Circuit for the first time affirmed the patentability of algorithms that merely manipulated data, even if only by applying well-known mathematical formulae, if the "invention as a whole" constitutes a novel, useful invention. The court did not expressly reject the Freeman-Walter-Abele analysis (indeed, it makes no mention whatsoever of the test), but clearly suggested that a deeper level of analysis for software patent applications was required than considering only the "pre-processing inputs" and "post-processing activity" as previously held.

Had the opinion ended here, the Federal Circuit's decision could be read as being limited to allowing patents only for *hardware* that operates solely to transform data. Alappat's application claims only a "rasterizer," which could be interpreted as a separate electric circuit that processes this data; the application makes no specific reference to software for achieving the same result. But the court went on to expand this change by including within its holding programs running on a general-purpose computer that only transform one set of data into another. It did so by making a novel observation: "[A] general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software."[33] This statement implies that software may be patentable not as a "process," which causes problems by conflicting with traditional restrictions on "mental processes," but as a "machine," since software executing on a general-purpose machine creates, in essence, a whole new machine that may be patentable when considered as a whole.

Chief Judge Archer wrote a scathing dissent in *Alappat* that may be captured by her statement: "What is going on here is a charade."[34] Unlike the majority, which merely

[33] *Id.*, at 1545.
[34] *Id.*, at 1564.

distinguished *Gottschalk* and *Diehr*, Judge Archer accused the court of reversing precedent and ignoring the physicality requirement that historically divided unpatentable mental processes from patentable machine inventions: "Where the invention or discovery is only of mathematics, the invention or discovery is not the 'kind' of discovery the patent law was designed to protect and even the most narrowly drawn claim must fail."[35] Because Alappat's invention dealt solely with the mathematical manipulation of data – "[t]he calculations are the beginning and end of the claim"[36] – Judge Archer determined that the whole of the invention constituted a "mathematical operation," and should be rejected under 35 USC § 101. Judge Archer concluded by warning that the majority opinion was "dangerous" because it allowed for patents on mathematical operations as long as they were recited in the structure of software, thus opening the door for a flood of patents on "bare mathematics" that had been rejected out of hand by the patent office (and the federal courts) for over 100 years.[37]

Nevertheless, the barriers against software patents continued to fall. The final blow to these restrictions arrived in the form of *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*,[38] in which the Federal Circuit considered the patentability of not only a software invention, but a software invention used to apply a mathematical formula that represented a business method. Despite the *three* separate objections to patentability, any one of which would historically have doomed the

[35] *Id.*, at 1557, *citing* Diehr, 450 U.S. at 192 n. 14.
[36] *Id.*, at 1564.
[37] *Id.*, at 1568.
[38] 149 F.3d 1368 (Fed.Cir. 1998).

application to certain death during the examination process, the Federal Circuit held the invention patentable, and the Supreme Court implicitly affirmed by denying certiorari.[39]

The invention forming the basis of this case involved a method for tracking individual mutual funds that have been pooled together into a single portfolio. Before the development of the claimed software, calculating the shares of each mutual fund contained in a merged portfolio on a daily basis was a laborious and time-consuming task, and further analysis of aggregate income and capital gains was extremely difficult. Such delays proved costly to investors, who often needed this information quickly. The software package developed by the appellants employed a method known as "Hub and Spoke" to track this information on a daily basis and perform these calculations and analysis very quickly. After obtaining a patent on the invention,[40] the appellant attempted to license the software to the appellee, a competitor; when negotiations broke down, the appellee sought declaratory judgment of invalidity.[41] The district court agreed, holding that the invention, on its face, "is an apparatus designed to solve a mathematical problem,"[42] that the invention failed the physicality element of the Freeman-Walter-Abele test and thus represented unpatentable software,[43] and fell prey to "the long-established principle that 'business "plans" and "systems" are not patentable, even though they not be dependent on the aesthetic, emotional, or judgmental reactions of a human.'"[44] On the basis of these three well-established principles, the district court granted summary judgment of invalidity.

---

[39] 525 U.S. 1093 (1999).
[40] U.S. Patent No. 5,193,056.
[41] 927 F.Supp. 502 (D.Mass. 1996).
[42] *Id.*, at 513.
[43] *Id.*, at 514.
[44] *Id.*, at 515, citing DONALD S. CHISUM, PATENTS: A TREATISE ON THE LAW OF PATENTABILITY, VALIDITY AND INFRINGEMENT § 103[5] at 1-75 (1990).

Surprisingly, on appeal, the Federal Circuit reversed on all three grounds. After setting forth the standard of review, the Federal Circuit first turned its attention to the argument that the invention represented only a machine implementation of a mathematical algorithm, and consequently was not patentable. The Federal Circuit held that the invention cleared this hurdle, stating: "[W]e hold that the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces 'a useful, concrete and tangible result' – a final share price momentarily fixed for recording and reporting purposes and even accepted and relied upon by regulatory authorities and in subsequent trades."[45] Next, the Federal Circuit reversed the district court's finding of invalidity on the basis of the Freeman-Walter-Abele test. The Federal Circuit expressly criticized this test, calling it "misleading," "confusing," and "ha[ving] little, if any, applicability to determining the presence of statutory subject matter."[46] The Federal Circuit proceeded, in essence, to eliminate the last vestiges of a categorical ban against patents for computer software:

> The dispositive inquiry is whether the claim as a whole is directed to statutory subject matter. It is irrelevant that a claim may contain, as part of the whole, subject matter which would not be patentable by itself. "A claim drawn to subject matter otherwise statutory does not become nonstatutory simply because it uses a mathematical formula, computer program or digital computer." *Diehr*, 450 U.S. at 187, 101 S.Ct. 1048.
>
> Every step-by-step process, be it electronic or chemical or mechanical, involves an algorithm in the broad sense of the term. Since [35 USC] § 101 expressly includes processes as a category of inventions which may be patented and § 100(b) further defines the word "process" as meaning "process, art or method, and includes a new use of a known process, machine, manufacture, composition of matter, or material," it follows that it is no ground

[45] 149 F.3d at 1373.
[46] *Id.*, at 1374.

> for holding a claim is directed to nonstatutory subject matter to say it includes or is directed to an algorithm.[47]

Finally, although the "business method exception" is beyond the scope of this discussion, it is noteworthy that the Federal Circuit harshly criticized the exception and provided a clear statement that no such exception would continue to survive:

> As an alternative ground for invalidating [State Street Bank's] patent under [35 USC] § 101, the court relied on the judicially-created, so-called "business method" exception to statutory subject matter. We take this opportunity to lay this ill-conceived exception to rest. Since its inception, the "business method" exception has merely represented the application of some general, but no longer applicable legal principle, perhaps arising out of the "requirement for invention" – which was eliminated by [35 USC] § 103. Since the 1952 Patent Act, business methods have been, and should have been, subject to the same legal requirements for patentability as applied to any other process or method.

It may be an understatement that the Federal Circuit's subsequent reversal on all three grounds came as a surprise, and the impact of this decision caused two fundamental changes in patent law. Not only did the Federal Circuit eradicate the "business method" exception, thus opening the gates to a flood of patent applications for business methods, but the Federal Circuit issued its clearest statement to the patent office that software patents are equally worthy of patents as any other invention, and that future rejections on the grounds that the software did not fall within a statutory class would no longer be tolerated.

This decision epresents the culmination in a steady relaxation of the reluctance of the courts to allow software patents; the wall built by *Gottschalk* first sprang a leak in *Diehr*, then a rivulet in *Alappat*, and was finally leveled by the Federal Circuit in *State Street Bank*. This trend would lay the foundation for the problems that were already

[47] *Id.*, at 1374-75.

forming when *State Street Bank* was adjudicated, and that would spring to full light in the coming years.

IV. THE EFFECT: CHAOS IN THE PATENT AND TRADEMARK OFFICE

The Patent and Trademark Office is responsible for processing an enormous number of patent applications; in 1999, the patent office received over 285,000 patent applications and issued over 169,000 patents.[48] In order to handle such a tremendous workload, the patent office procedures are carefully organized and have been honed over the course of 160 years of operation.[49] The patent office employs several corps of examiners, each specially trained in a different set of scientific arts, such as electrical engineering, chemistry, or biotechnology, and new applications received by the patent office are forwarded to the most appropriate examining group.[50] After an initial reading of the application to get a general sense of its subject matter, the examiner first researches the relevant areas of science by turning to an established library of "prior art" documents maintained by the patent office.[51] This library consists of issued patents, abandoned and pending patent applications, publications, and general textbooks on the subject; furthermore, these documents are carefully categorized by subject matter in order to focus the examiner's research on those inventions most relevant to the classification of the claimed invention. In so researching, the examiner discovers the state of the industry

[48] Patent applications received include 270,187 utility patent applications, 14,732 design patent applications, and 421 plant patent applications. This data is provided by the U.S. Patent and Trademark Office website at http://www.uspto.gov/web/offices/ac/ido/oeip/taf/reports.htm .
[49] The current U.S. Patent and Trademark Office was created with the passage of the Patent Act of 1832.
[50] *See* MANUAL OF PATENT EXAMINING PROCEDURE § 504; *see generally* "Examination Guidelines for Computer-Related Inventions," *available at* http://www.uspto.gov/web/offices/com/hearings/software/analysis/computer.html .
[51] *See* MANUAL OF PATENT EXAMINING PROCEDURE § 704.

prior to the claimed invention and uses this determination to decide the novelty of the invention, which is critical to patentability.

These procedures work well when all of the essential resources – a cadre of experienced examiners, a substantial body of prior art, and a hierarchical organization of the library – are firmly established. Without such carefully organized resources, the patent office simply could not handle patent applications efficiently and effectively. However, just such a scenario arose when the federal courts' reversal on the patentability of software inventions caught the patent office unprepared.[52] The patent office found itself in the uncomfortable position of being required by court decisions to examine a sudden glut of software patent applications but being unequipped to do so.

The result of this difficult position was that software patent applications were sent to examiners untrained in software engineering and lacking in any useful prior art library. The examiners lacked the knowledge of the software field and industry to recognize an anticipated or obvious invention, and they could not find prior art to oppose the claims. For lack of a sufficient basis for a rejection, the examiners were forced to allow many patents for software inventions of dubious utility, novelty, and non-obviousness without the probing inquiry associated with applications in other fields. Thus, questionable (and, in some cases, obviously invalid) software inventions frequently received patent protection.

This situation was hardly unanticipated by patent law scholars. One of the principal arguments that the patent office set forth against allowing software patents was

[52] *See*, KENNETH NICHOLS, INVENTING SOFTWARE: THE RISE OF "COMPUTER-RELATED" PATENTS 102 ("It is widely acknowledged that the apparent policy reversal by the courts caught the PTO unprepared, with no cadre of examiners trained in computer science, no suitable classification scheme, and no prior-art search capability.")

that it lacked the resources to examine such applications effectively.[53] The *Gottschalk* Court's rejection of software patents was based in part on a 1966 report of a Presidential Commission that warned:

> The Patent Office now cannot examine applications for programs because of a lack of classification technique and the requisite search files. Even if these were available, reliable searches would not be feasible or economic because of the tremendous volume of prior art being generated.[54]

More recently, Chief Judge Archer's dissent in *Alappat* warned that the continued relaxation of the barriers to software patents could lead to a "dangerous" flood of applications for purely mathematical software inventions.[55] Nevertheless, the *Alappat* majority and subsequent decisions steadily eliminated the barrier, creating a number of problems for the patent office.

A. CURRENT PROBLEMS WITH SOFTWARE PATENT EXAMINATION

The challenges faced by the patent office in evaluating software patent applications may be boiled down to four basic problems: A surge in the number of software patent applications, the lack of examiners skilled in software engineering, an insufficient prior art library (combined with a general failure of software patent applicants to meet their duty of disclosing relevant prior art), and the lack of a workable classification scheme for software inventions and prior art.

---

[53] G. GERVAISE DAVIS III, ESQ. SOFTWARE PROTECTION: PRACTICAL AND LEGAL STEPS TO PROTECT AND MARKET COMPUTER PROGRAMS. 155 (Van Nostrand Reinhold Co. Inc. 1985) ("The Patent Office and the CCPA, as the then principal court for patent appeals, engaged in a tug-of-war for a few years over software patentability. The Patent Office's position was that it did not have the facilities to deal with software patents, so it did not want software to be patentable.")

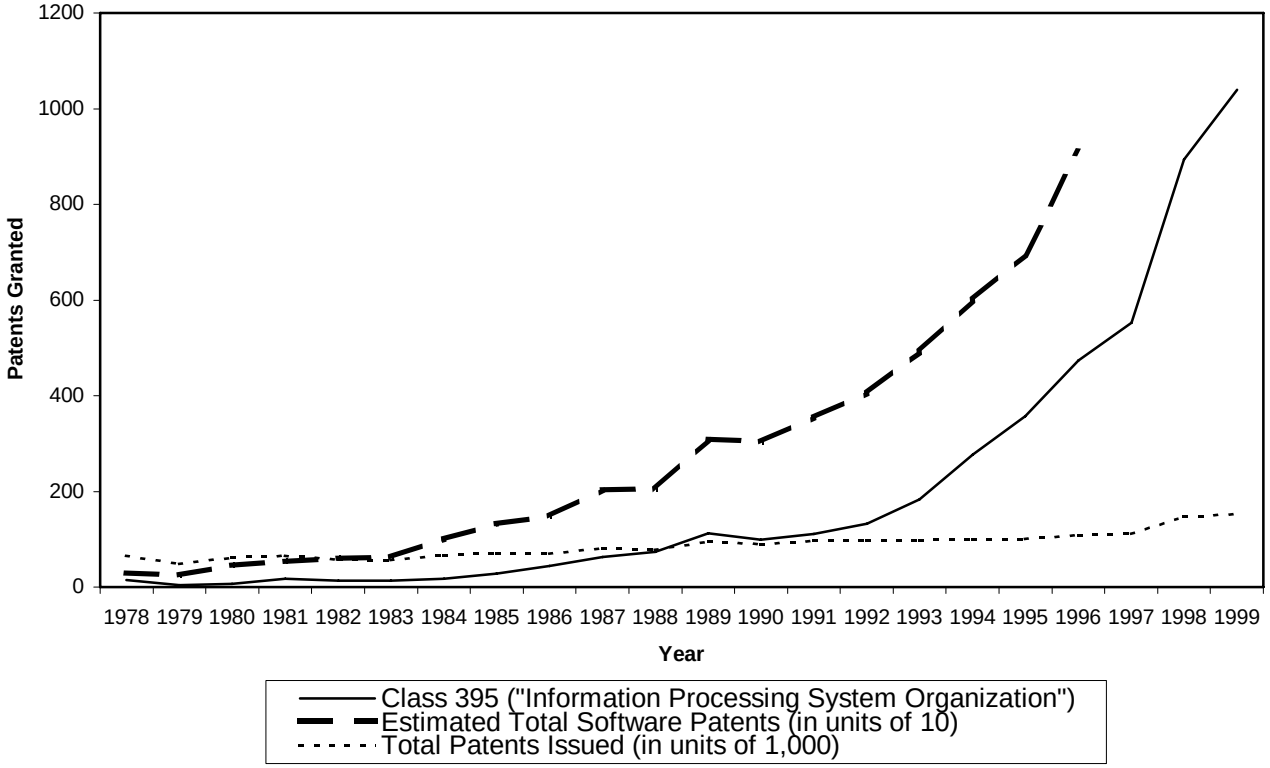
[54] "To Promote the Progress of … Useful Arts," REPORT OF THE PRESIDENT'S COMMISSION ON THE PATENT SYSTEM (1966), *reproduced in* Gottschalk v. Benson, 409 U.S. 63, 72 n. 4 (1972).

[55] In re Alappat, 33 F.3d 1526, 1568 (CAFC 1994).

First, in response to federal courts' encouragement of software patents, the number of software patent applications received annually by the patent office has grown considerably. Although difficult to measure, the number of software patent applications being submitted is reflected in the number of software patents issued, which has exhibited explosive growth in the past decade, with estimates placing the number of software patents issued in recent years between 6,000 and 7,000.[56] Even given a 38% increase over the past eight years in annual submission of *all* types of patent applications,[57] the exponential increase in the number of *software* patent applications far outpaces this figure.[58] Clearly, the patent office is struggling to cope with a flood of software patents – and, at the same time, has experienced a reduction in operating capital.[59]

[56] *See*, Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," at 3, *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 280-81 (1996); Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html .
[57] Philip E. Ross, "Patently Absurd," *Forbes* May 29, 2000, *reproduced at* http://www.forbes.com/forbesglobal/00/0529/0311090a.htm ("In just eight years the number of patent applications is up 38%, to 290,000 a year.")
[58] *Id*.
[59] *See, id.* ("The patent bureaucracy's budget hasn't kept pace with the exploding workload. From fiscal 1998 to 1999, patent applications rose 11%, but the office endured a 16% decline in how much fee revenue it was allowed to spend on operations.")

Figure 1. Number of Software Patents Granted Compared to Total Number of Patents Granted



Second, the examiners currently employed by the patent office are not well versed in the software industry and state of the art. The patent office has failed to recognize software as a separate field worthy of its own examining corps.[60] This decision led to an inadequate examination of software patents by examiners trained in other areas of

[60] *See*, Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 284-85 (1996) ("Traditionally, the patent examining core was divided in the three examining areas: mechanical, electrical, and chemical. With the advent of the increased patenting of inventions in the biotechnology area, Examiners which have experience in biotechnology were hired to handle the large number of biotechnology applications on file. Software, however, has not been embraced in the same way as biotechnology because of the way that software was regarded – the "stepchild" of hardware; and as such, is not considered a science in and of itself.")

science, particularly by electrical engineers.[61] The first batch of computer science graduates hired for the primary purpose of examining software patents were accepted as late as mid-1994; furthermore, only ten were hired, and they were required to spend approximately two years training for the position.[62] Their main resource for learning the particulars of the examination of software patent applications is the "Examination Guidelines for Computer-Related Inventions," which are, in a word, unhelpful.[63] The general consensus is that patent examiners are less competent in evaluating software inventions than other kinds of patentable inventions.[64] The patent office apparently agrees; it recently adopted a rule that requires web-oriented patents, which have drawn particularly heavy public outcry over their lack of novelty and non-obviousness, not only to be approved by a regular examiner, but also must survive a second level of review by a senior patent examiner.[65]

---

[61] *See*, Kennard, *id.*, at 8,*reproduced in* BROWN, *id.*, at 280-81 ("The Examiners who the Patent Office has given the responsibility to handle the prosecution of software-type applications are the ones who handle hardware-type (electrical and electronic circuitry) applications. These examiners, for the most part, may have a sound understanding of hardware, but software is a different technology that many of them may not be particularly equipped to handle.") *See also*, LEMLEY, MENELL, MERGES, & SAMUELSON, SOFTWARE AND INTERNET LAW 104 (2000) ("[U]ntil recently, the Patent Office has only recently taken steps to hire patent examiners qualified in computer software or related fields. During the 1980s and the early part of the 1990s, the flood of software patent applications were handled largely by people operating outside their area of expertise.")

[62] *See*, BERNARD A. GALLER, SOFTWARE AND INTELLECTUAL PROPERTY PROTECTION: COPYRIGHT AND PATENT ISSUES FOR COMPUTER AND LEGAL PROFESSIONALS 36 (1995).

[63] The patent office's "Examination Guidelines for Computer-Related Inventions," available at http://www.uspto.gov/web/offices/com/hearings/software/analysis/computer.html , essentially rephrase the general procedure of examination, with very little guidance specifically relevant to software. Furthermore, even the "final version," released on February 28, 1996, is not entirely reliable; for example, it recommends the *Diehr* test, consisting of analysis of any "pre-computer process activity" and "post-computer process activity," that was expressly disavowed in *Alappat*. Any rejection based on the test advocated by the guidelines could well be reversed – in fact, this is a concise description of the holding in *Alappat* itself.

[64] Philip E. Ross, "Patently Absurd," *Forbes* May 29, 2000, *reproduced at* http://www.forbes.com/forbesglobal/00/0529/0311090a.htm .

[65] *See, id.* ("Todd Dickinson, who heads the patent office, acknowledged the problems [with software patents] recently when he announced that web-oriented patents would have to clear both a regular examiner and a new layer of scrutiny by a more senior official.")

Although any veteran examiner can apply the same procedures of examination process to software patents, this lack of specialized knowledge of the software field hampers the ability of examiners to analyze software patents in several ways. Because the workload carried by an examiner limits the amount of time he can spend on any single application, a more experienced examiner need not waste time learning the basics of the field and the general state of the art and can spend more time on prior art research. Furthermore, examiners who have more formal education and practical familiarity with software are more likely to recognize common, "anecdotal" examples that constitute the foundation of software development and better able to stand in the shoes of "one of ordinary skill in the art" in order to make sound, reliable rulings on matters of obviousness and novelty.[66] Finally, examiners who understand the nature of software are more capable of judging whether the claimed invention is merely a software analogue of a well-known process. The public outcry against software patents that merely implement an old function in a new form[67] can be addressed by enlisting more patent examiners with extensive knowledge of software.

[66] This lack of familiarity with "anecdotal" evidence is hard to dispute in light of some patents that have been granted. One telling example is U.S. Patent No. 5,937,438, granted to Lucent Technologies for a "sine-cosine lookup table." A computation-intensive computer algorithm can be optimized by eliminating real-time computation of sine and cosine values: the values can be computed once, stored in memory as a table, and referenced when these values are needed. This method is, perhaps, one of the most basic, common tools in the software developer's mental toolkit. It has been taught in virtually every computer science curriculum for decades, and even implemented on several occasions by this author, admittedly several steps short of being one of "ordinary skill in the art." Any software developer that would deny a prior knowledge of this method, or at the very least could not design such a table with little prompting, would raise serious questions of his own competency or "skill in the art." It is surprising that Lucent would be brazen enough to assert that its development of this method constitutes "invention," and it is nothing short of astonishing that the patent office would agree.

[67] Although patentable inventions require evidence of a demonstrable invention (e.g., evidence that a machine invention has been properly reduced to practice by constructing and testing a prototype), a patent is aimed at the underlying principle of the invention. Consequently, in theory, the patent office will not allow a patent that implements a previously known method in a new form (e.g., one patent for carrying out a process in hardware and another for performing the same process in software.) However, heavy criticism is aimed at software patents for doing just this.

Third, examiners, skilled and unskilled alike, are hindered by the lack of an adequate prior art library. As described above, the first step in the examination process (after reading the patent application) is an extensive search of the prior art to find any references that might render the claimed invention anticipated or obvious. Such references instruct the examiner about the state of the art and help the examiner to decide whether to issue a rejection or an allowance. Furthermore, since most rejections must be based on references,[68] an inability to supply a prior art references makes the examiner more likely to issue an allowance instead of an untenable rejection. Consequently, the inadequacy of a software library would, in theory, lead to fewer and weaker rejections, and therefore a higher rate of patents issuing that are invalid or at least questionable. As will be demonstrated, the issuance of many undeserving patents is exactly the criticism leveled at the patent office over software patents.

Much has been written about the inadequacy of the patent office's prior art library for the software field.[69] In fairness, much of the problem stems from two factors that are

[68] The most frequent causes for rejection, and the most frequent subject of debate between the examiner and the applicant, are anticipation and non-obviousness. In these cases, the only occasions in which a rejection may be made without citing a reference is when the examiner can rely on personal knowledge of the prior art, or when an admission by the applicant warrants a rejection. Bust in most cases, the rejection is based on prior art references, in which case "the examiner must cite the best references at his or her command." MANUAL OF PATENT EXAMINING PROCEDURE § 707(c)(2).

[69] *See,* U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE AND INTELLECTUAL PROPERTY – BACKGROUND PAPER 8-9 (Global Press Publications, 1990) ("One of the these problems is the incomplete stock of 'prior art' available to patent examiners in evaluating patent applications for processes involving computers, especially those involving software and algorithms."; "[T]he available and locatable prior art [for software] is less complete and relatively more difficult to compile or search than the prior art for other technical fields."); *id.*, at 22 ("Many businesses and software industry members time and time again have stated the following: *I do not know how the Patent Office could issue that patent in light of the literature, a thesis, or other non-patent documents of which I am aware!* As such, prior art, particularly in software patenting, looms as one of the most important issues."); Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("Prior art [for software] continues to be a disaster area at all of the patent offices around the world. Being issued a software patent means absolutely nothing with regards to novelty and obviousness, as examiners continue to be provided with only about 25% of the relevant prior art. Examiners do not have the time, resources and search training to complete the tasks they are charged with…."); U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL

beyond the control of the patent office. First, prior art is more difficult to locate and understand for software than in other industries because of software developers' habitual reliance on trade secret protection.[70] In many industries, when the inventor sells his product to the public, he creates the opportunity for competitors to reverse-engineer the invention in order to understand and copy it; but for software, reverse-engineering is very difficult and speculative without the benefit of the source code. Therefore, withholding the source code is an easy and effective method of employing trade secret protection – but this secrecy also hampers the ability of patent examiners to assess prior art and the general state of the industry. As a result, the only large sources of prior art are scattered journal articles and previously issued software patents. This problem is evidenced by observations that the typical software patent cites very few references that are not other software patents (far fewer, in fact, than for most academic publications[71]), and that half of all software patents cite *only* other software patents as prior art.[72]

CHANGE 23-24 (Global Press Publications, 1993) ("In addition, the PTO has an incomplete database of "prior art" for software-related inventions. This makes it even more difficult for examiners to judge whether an application describes a "novel" and "nonobvious" invention."); *id.*, at 23-24 fn 89 ("Computer processes are not classified within USPTO's patent classification system in any readily identifiable set of classes and subclasses."); Wayne Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* Peter Brown, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 290-91 (Practice Law Institute, 1996) ("If a random selection of software patents is reviewed, it will be quickly noted that the average number of prior art references cited in each is small. In fact, statistically, you would find that the average number of references cited is approximately 11 references per patent.")

[70] *See,* U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE AND INTELLECTUAL PROPERTY – BACKGROUND PAPER 22 (Global Press Publications, 1990) ("Because the bulk of software continues to be protected by copyright and/or trade secret, because much of the history of software development is not in the published literature, and because relatively few patents for software-related inventions were granted prior to the 1980s, the available and locatable prior art is less complete and relatively more difficult to compile or search than the prior art for other technical fields."); U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 8 (Global Press Publications, 1993) ("Among the reasons cited for the perceived problem of prior art is the extensive use of trade secret protection for computer software. … [T]he resources available to the PTO for determining obviousness and novelty are more limited than in other fields; in large part the prior art database is limited to software that is already the subject of other patents for software-related inventions.")

[71] *See, id.* ("In 1995, the average software patent cited 2.1 non-patent prior art items, up from 1.9 non-patent prior art items in the first half of 1994, a change of 0.2 items (for 1000+ sample sizes). … During

Additionally, patent applicants and practitioners undertake a general duty to disclose to the examiner all relevant prior art known to the applicant,[73] but have been shirking this duty.[74] This is particularly detrimental to the examination process because one would expect that, when faced with the problem that prompted the software invention, the developer has spent considerable time and money investigating the alternatives, and therefore is in the *best* position to provide prior art examples. However, if the applicant cites few or no prior art references, the examiner has fewer grounds for objection to the application; this expedites the prosecution process and results in a broader patent than one that is hemmed in by limitations based on prior art. And since reports of discipline for failure to fulfill the duty of disclosure are sparse, one can only conclude that the patent office's growing workload and shrinking budget are contributing to the inability of the patent office to enforce this duty and punish offenders. Faced with several incentives not to cite prior art and no compulsion to do so, it is unsurprising that applicants are citing very little prior art,[75] providing fewer reasons for the examiner to limit or reject the application.

---

this time period, the average IEEE/ACM article had over 30 citations in the bibliography, growing at a rate of 0.9 items per year. Thus not only are software patents still citing a small amount of materials as compared to journal articles covering these same topics, but the rate of growth of software patent citations is much slower than that of journal bibliographies.")

[72] *See*, Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("Half (50%) of all issued software patents cite no non-patent prior art.")

[73] *See generally*, MANUAL OF PATENT EXAMINING PROCEDURE Chapter 2100; 37 CFR 1.56(a) ("Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability….")

[74] *See*, Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("Since anyone with a pulse by now recognizes that examiners have neither the time, resources, tools and library science expertise to conduct detailed prior art searches for all the software patents crossing their desks, applicants are exploiting the lack of a formal requirement to send in any prior art.")

[75] Despite the short-term incentives, this tactic may well be more costly in the long run. Patent rights are more likely to be enforced by courts and better respected by competitors if they are clearly defined, apparently meritorious, and expressly distinguished from prior art examples, than those that are broad but

The fourth and final problem with software patent examination is that the current classification system used by the patent office to sort prior art references is poorly developed for the software field. The patent office organizes its prior art library by setting forth classifications of invention that group together similar references. These classifications are well-defined for traditional fields of invention, such as electrical and mechanical engineering, but very few classifications have been established for software.[76] Even Class 395, entitled "Information Processing System Organization," which by default has received the bulk of software prior art examples, is not limited to software (it contains several subclasses for hardware), and even the use of this class is declining.[77] While the likely reason for not establishing specific classes for software may be that such a division would hinder a comparison with hardware analogues of the software invention, the tradeoff is an obstacle to comparing several software inventions, which are now scattered throughout the classification system.[78] Finally, it is surprising to note that

---

vague, apparently meritless, and probably invalidated by the prior art. Although the latter patent is easier to obtain, the former may be easier to enforce against infringers and less likely to be held invalid; therefore, the benefits of quicker, cheaper prosecution may be overwhelmed by the increased likelihood and cost of litigation and the greater chance that the patent will be invalidated. Therefore, the applicant may have an incentive to cite prior art references and cooperate with the examiner to obtain a stronger patent. *See*, Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 291 (Practicing Law Institute 1996) ("Many software patents have issued without the most pertinent prior art having been considered by the Examiner. This is something that the software patent applicant should want to avoid and should work actively to prevent so that the patent that issues from an application will be far less likely to held invalid in view of prior art.")

[76] The only classification that can readily be recognized to contain only software inventions is Class 717, which contains "Data Processing: Software Development, Installation, and Management"; but, of course, these tools comprise only a small fraction of software inventions.

[77] *See*, Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("364 and 395 are the two main computing classifications at the PTO for computer hardware and software. For the first year ever, the number of software patents exceeded the number of patents so classified, with the statistical trend of software patents continuing to reflect a different process than the trend for patents classified 364/395.")

[78] *See*, U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE AND INTELLECTUAL PROPERTY – BACKGROUND PAPER 9 (Global Press Publications, 1990) ("Another problem is the lack of special classifications or cross-references to issued patents. As a result, it is virtually impossible to find, let alone count or profile, all software-related or algorithmic patents. This means that patent examiners and the

despite the close involvement of the patent office with technology, the patent office's prior art library provides only the most basic computerized access; simple functions like cross-referencing can only be conducted manually; and examiners access and search the prior art library primarily by hand.[79] Thus, even if prior art examples were readily available and patent applicants cited more prior art references, it is foreseeable that this availability would only exacerbate the problem by dumping even more references into a badly-organized prior art database. A compelling sign of the impact of this poor organizational scheme is that the patent office cannot even count the number of software patents being issued.[80]

B. CONSEQUENCES OF PROBLEMS WITH SOFTWARE PATENT EXAMINATION

The Framers of the Constitution wanted to promote innovation and protect the rights of inventors, but many – including Thomas Jefferson, the main proponent of

---

public have no effective way of searching and studying such patents."); LEMLEY, MENELL, MERGES, & SAMUELSON, SOFTWARE AND INTERNET LAW 331-32 (2000) ("[T]he PTO's classification system is not equipped to handle software patents. As a result, software patents tend to be classified according to the field in which the software will ultimately be used, rather than according to the nature of the software invention.")

[79] *See*, U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 24-25 (Global Press Publications, 1993) ("Despite the intense controversy and policy focus on these areas of art since *Diamond v. Diehr*, PTO reported to OTA that it has no provisions for flagging, crossreferencing, or otherwise efficiently monitoring and reporting prosecution, issuance, and litigation for these types of patents, except through time-consuming manual search, review, and selection from various large patent subclasses."); LEMLEY, MENELL, MERGES, & SAMUELSON, SOFTWARE AND INTERNET LAW 331-32 (2000) ("[T]he PTO's classification system is not equipped to handle software patents. … This makes it much harder for examiners to find what prior art exists, a problem that could also be solved, either by reclassification or by an increased emphasis on computer search systems.")

[80] *See*, U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 24-25 ("OTA also finds that improving electronic search and retrieval capabilities for the PTO's own database is critical, because it is used by the PTO's own examiners during the application process and by the public. In September 1991, PTO reported that it is unable to provide statistics on the number of patents issued for software-related inventions (e.g., patents for computer processes and algorithms), which PTO refers to as 'computer-implemented process patents.'")

patents and, in essence, the first Commissioner of the patent office – expressed hesitation to reward patentees with short-term monopolies.[81] Their trepidation was quelled by establishing patent examiners who represent the public interest in the *ex parte* patent prosecution process.[82] Although the public, competitors of the applicant, and industry experts usually do not learn of a patent application until it issues as a patent,[83] the rights of these interested parties are assumed to be represented by the patent examiner during prosecution. Because of this presumption, an issued patent carries a court-enforced presumption of validity, and accused infringers bear the evidentiary burden of rebuttal.

With software patents, however, it is arguable that this model has broken down. The substantial barriers to effective patent examination have led to a tremendous number of issued software patents and harsh criticism by the public that many of these patents are invalid. Critics of software patents frequently cite a stream of patents that clearly should have been rejected out of hand by a competent examination system, such a claim for any sort of window-based operating system, a claim for a sine/cosine lookup table, and a

[81] Letter from Thomas Jefferson to David Humphreys of 6/23/1791, *reproduced in* P.L. FORD, THE WORKS OF THOMAS JEFFERSON 272-73 (1904), *reproduced in* Edward C. Walterscheid, *Patents and Manufacturing in the Early Republic*, 80 J.PAT. & TRADEMARK OFF. SOC'Y 885, 859 (1998) ("[O]ther nations have thought that these monopolies produce more embarrassment than advantage to society; and it may be observed, that the nations which refuse monopolies of invention, are as fruitful as England in new and useful devices."

[82] The current practice of examination only dates back to the creation of the patent office in 1836. However, the Framers originally implemented an examination process with the Patent Act of 1790; under this act, patents were granted only with the approval of the Secretary of State, the Secretary of War, and the Attorney General. Unfortunately, the number of patent applications swamped these officers and detracted from their other duties, and the young government could not afford a specialized body for assuming these responsibilities; therefore, the examination process was reluctantly abandoned with the Patent Act of 1793 in favor of a registration system. By all accounts, the registration system proved disastrous and was replaced in 1836, when an affordable examination system was devised. Despite this brief abandonment of examination, it is apparent that the Framers of the Constitution and the original drafters of the Patent Act intended to establish an examiner to protect the interests of the public. *See generally*, BRUCE W. BUGBEE, THE GENESIS OF AMERICAN PATENT AND COPYRIGHT LAW (1967).

[83] The patent office maintains all applications in strict confidence until a patent issues. These rules have only recently been changed to comply with international practices, such that patent applications are published after 18 months of pendency, but during the critical period of examination this secrecy is preserved.

claim covering all Internet search engines.[84] Some patents, such as Amazon.com's "1-Click" patent,[85] have also drawn a great deal of mainstream public objection and even a public boycott of the company. The public confidence in the patent system has been damaged as a result of these problems.[86]

---

[84] *See, e.g.,* BERNARD A. GALLER, SOFTWARE AND INTELLECTUAL PROPERTY PROTECTION: COPYRIGHT AND PATENT ISSUES FOR COMPUTER AND LEGAL PROFESSIONALS 33-34 (Quantum Books 1995):

> In November 1993, Compton Encyclopedia, Inc., a subsidiary of The Tribune Company, announced that it had obtained a patent covering some basic techniques for searching and retrieving multi-media information from CD-ROM databases, and that it intended to enforce the patent by asking for royalties from licensees. An immediate uproar ensued, with many people coming up with evidence that they argued was prior art, which should invalidate the claimed invention. At issue was the future of a new field of application in the software field, a field that is currently one of the most active in the industry.
>
> Bruce Lehman, commissioner of Patents and Trademarks, quickly ordered a reexamination of the Compton patent. … Subsequent to this order, in March 1994, the PTO issued a news release stating: "All claims in Compton's multi-media patent issued in August 1993 have been rejected on the grounds that they lack 'novelty' or are obvious in view of prior art."

See also, id., at 36-37:

> On February 21, 1989, Cary L. Queen was awarded Patent #4,807,182, which he had applied for on March 12, 1986. The patent was then assigned to Advanced Software, Inc. This patent describes a process for comparing two documents and identifies words and sentences that are different between them. …
>
> Software for comparing two files, or bodies of text, has existed for many years. …
>
> What is introduced as novel in this patent application is the specific way the comparison function is applied to documents produced by word-processing software. ...
>
> The author of this patent added to the usual comparison function the ability to consider an entire paragraph at a time …
>
> This was the novel idea of the patent, and the PTO decided that it was indeed novel. One might question, however, whether it might have been obvious to someone outside the PTO who was skilled in the art. It is the kind of thing that would strike a computer programmer as obvious. In fact, most inventions are probably of that kind, but someone is the first to identify them as worth pursuing and developing. …

See also, U.S. Patent No. 5,715,314 (a patent for "network-based sales systems" – in essence, this patent claims *all e*-commerce); U.S. Patent No. 5,937,438 (a patent for "sine/cosine lookup table" – in essence, this patent claims an *extremely* common, basic, and long-practiced software tool, as discussed more fully *supra* at footnote 66); U.S. Patent No. 4,555,755 (a patent for "windowing" – in essence, this patent claims any sort of "window" display); U.S. Patent No. 4,355,371 (a patent for "instantaneous alpha content prescan method for automatic spelling error correction" – in essence, this patent claims a spell-checking in a word-processing application); U.S. Patent No. 4,308,582 (a patent for "precusory set-up for a word processing system" – in essence, this patent claims the use of a menu system in a word-processing application); and 38 patents owned by AltaVista, Inc. that fundamentally cover Internet-based search indices and are being used as the basis for threats to eliminate every other web-based search engine, including Yahoo, Excite, Lycos, and Infoseek, as infringers.

[85] U.S. Patent No. 5,960,411.

[86] *See*, BERNARD A. GALLER, SOFTWARE AND INTELLECTUAL PROPERTY PROTECTION: COPYRIGHT AND PATENT ISSUES FOR COMPUTER AND LEGAL PROFESSIONALS 31(Quantum Books 1995) ("The concern is that software may be patented by someone who will then suddenly start demanding royalties from those who are freely using techniques long considered to be in the public domain. Behind these concerns is a

Because software patent applications are not adequately examined and many software patents are of questionable merit, it is unclear whether the assumptions that underlie the examination process hold true for software patents: are the interests of the public and the industry adequately protected by examination, and should issued patents carry a presumption of validity? Some critics of software patents argue that the current system is little more than a "patent registration system," such as that instituted by the Patent Act of 1793, whereby patents were granted automatically upon filing a complete application but did not carry the customary presumption of validity. Indeed, the problems associated with software patents – a vast number of facially invalid patent applications, an examination process that barely rises above a cursory review of the application, a flood of issued patents that are demonstrably meritless and often granted for long-standing and well-known practices, and rampant abuse by patentees – also plagued the patent registration system and led to its eventual failure and abandonment.[87]

In addition to inadequate examination, software patents face more practical problems. Patent prosecution is very expensive: the average cost of preparing, filing, and prosecution an application to completion ranges from $20,000 to $50,000, with some instances reaching as high as $120,000.[88] These costs can only be tolerated by the largest software developers, thus acting as a barrier to entry, and are only justified under a good

belief that the PTO does not have enough expertise in software to recognize old and well-known ideas that have been around for years and have been freely used by everyone.")

[87] *See generally*, JOHN RUGGLES, REPORT UPON THE PATENT LAWS, REGISTER OF DEBATES, 24th Cong., 1st Sess. (1836).

[88] *See*, Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 292 (Practicing Law Institute 1996) ("Because of the disparity in the scope of software applications, the actual preparation costs for preparing software applications can range from as low as $7000.00 to more than $100,000.00. However, the average costs range for preparing software applications is from $10,000.00 to $30,000.00. The average costs for prosecuting the application before the patent office is from $10,000.00 to $20,000.00.")

expectation of substantial profit.[89] Additionally, patent prosecution typically takes up to two years to complete – an interminable period that may well outlast the lifetime of the software. Because of this long period, many software patents may not issue until well into the competitors' development process, leading to the problem of "landmine patents" (discussed below), or may have to be litigated after the infringing product has come and gone from the marketplace. Finally, patent applications must describe the invention in such detail that "one of ordinary skill in the art" can make and use the invention, which necessarily defeats trade secret protection.[90] Because trade secret protection for software is easy to implement, effective, costs nothing, takes effect immediately, and does not expire (absent independent development, industrial espionage, or voluntary surrender), many software inventors continue to rely on it as an alternative to costly, protracted, risky, and occasionally unenforceable software patents.

IV. THE RESULT: THE IMPACT OF PROBLEMATIC SOFTWARE PATENTS ON THE SOFTWARE INDUSTRY

A. THE REALITIES OF SOFTWARE PATENT ENFORCEMENT AND DEFENSE

The impetus for granting patents in a field of science is twofold: inventors benefit by acquiring temporary, but governmentally enforced, property rights in their inventions, thereby encouraging innovation; and the American people benefit from a full, public disclosure of the invention by the inventor and a dedication of the invention to the public

[89] But enforcement presents a new set of problems, discussed *infra*.
[90] One wonders, however, if this is true in light of the inexperience of software patent examiners and the cursory examination process, especially in light of the absence of a requirement that the application include source code. Is it possible that vague description of an invention could earn a patent, while being so vague that competitors cannot reasonably "make and use" the claimed invention?

domain upon the lapse of patent protection. The software industry has not realized these benefits from the federal courts' allowance of software patents. Certainly, the industry has changed to account for this new form of software protection, but few claim that the industry is materially enhanced by the introduction of software patents.

The realities of the present system of software patents make enforcement difficult. The closed-source nature of most commercial software and industry reliance on trade-secret protection makes detection of infringement difficult: an infringing algorithm can operate solely on the underlying operation of the software and have little or no impact on the observable behavior of the program. Even when it appears likely that infringement is occurring, the patentee may be hesitant to litigate due to the prohibitive cost (the average patent trial costs $1.2 million in legal fees) and unpredictable nature of patent litigation, with the worst-case outcome that the expensive patent will be invalidated and, having made a public disclosure of the invention, the patentee will have paid a substantial fee to surrender *all* protection of the invention. When combined with the facts that a patent can only be enforced after it has been granted,[91] and that prosecution often takes so long that it issues after the accused product has already been developed and marketed to its full extent,[92] enforcement of software patent rights becomes a very tenuous and dangerous practice.

But these problems cut both ways. Software development has become a risky prospect because being treats of infringement raise their own difficulties. Since patents

[91] *See*, G. GERVAISE DAVIS III, ESQ. SOFTWARE PROTECTION: PRACTICAL AND LEGAL STEPS TO PROTECT AND MARKET COMPUTER PROGRAMS. 161 (Van Nostrand Reinhold Co. Inc. 1985) ("[N]o protection is available until a patent has been issued….")

[92] *See*, BERNARD A. GALLER. SOFTWARE AND INTELLECTUAL PROPERTY PROTECTION: COPYRIGHT AND PATENT ISSUES FOR COMPUTER AND LEGAL PROFESSIONALS 31 (Quantum Books 1995) ("Even when an invention is really new and deserves patent protection, the procedure is secret and takes so long – years in most cases – that others are likely to come up with the same invention and find out only after several years of development and marketing that they have been infringing on someone else's patent.")

are being granted for a wide range of long-standing and common software practices, software developers may infringe many patents in the development process that it should reasonably expect to be in the public domain; and because the patent office's prior art library is meager and poorly organized, developers cannot find prior patents even if they search. Furthermore, because of the long duration of patent prosecution and the confidentiality of pending applications, developers could be faced with an accusation of infringement after they are far along in the development process. Such developers may be faced with the undesirable choice of paying a licensing fee for using an algorithm that it could not have predicted would later be patented, or retooling their product so extensively that it may be very costly or simply impossible. Even if the patent is facially invalid, the accused infringer must pay the legal fees of litigation – which, again, average $1.2 million – and risk a verdict of infringement, resulting in exorbitant damages and possibly an injunction. Thus, software developers may be hard pressed simply to pay the license fee than to challenge the patent.

This highlights a fundamental problem with software patents: Not only are many invalid software patents being granted, but invalidating them through litigation is extremely costly to an accused infringer and presents no tangible benefit over paying a license fee. The prior art library for software is being filled with patents that are incorrectly granted are not being invalidated after issue, rendering any prediction of which practices a developer may safely use hazardous at best.

B. THE CONSEQUENCES: CHANGES IN THE SOFTWARE DEVELOPMENT INDUSTRY

The consequences of these realities to the software industry are far-reaching and serious. A study of industry practices reveals that patents are not being used as the federal courts had expected.

First, the most common use of software patents is defensive patenting. Large developers, including Microsoft, IBM, and Apple, are filing for and receiving software patents in large quantities and filing many of them away for a "rainy day."[93] Because the consequences of an infringement threat or litigation can be severe, these developers are acquiring libraries of patents in order to use them against attacking patentees: they can counter threats of infringement with assertions of infringement of their own patents (and because these patents cover some very common practices, it is quite likely that any software project violates at least some of the many patents held by these companies.) The end result is a cross-licensing agreement, and this is common practice in the industry.

This result is undesirable for three reasons. First, cross-licensing defeats one of the fundamental purposes of software patents. No innovation is being protected by these patents; rather, large companies are formulating cooperative agreements that result in a negation of patent rights against each other. It is questionable whether this "tit-for-tat" wholesaling of patent rights bears even a speculative connection to promoting innovation in the software industry.

Second, defensive patent libraries can be used equally against assertions of infringement of both frivolous software patents *and* meritorious software patents. The owner of the patent for RSA algorithm has no greater leverage than the owner of a patent

[93] *See*, Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 299 (1996) ("Many large, medium, and small companies obtain software patents for a *rainy day*. The rainy day being referred to is when the company has an action brought against it for patent infringement. The software patents of the company's patent portfolio can be used for settlement and cross-licensing purposes.")

for a sine/cosine lookup table; both are patents that neither the patentee nor the accused infringer wants to litigate. Therefore, companies could essentially become infringement-proof by acquiring such a broad library of patents, both meritorious and frivolous, that no one can challenge them with infringement for fear of facing a dozen counterclaims for infringement of various patents.

Third, because developers need to license the rights to defensive patent libraries from a number of developers in order to develop *any* software safely, the cost of these license agreements creates a barrier to entry. New developers and small development companies often cannot afford these license fees and are being excluded from the industry. In addition, developers must retain a patent attorney in order to find and avoid infringing patents "at every stage of the development process." It is likely that a study of the software industry would reveal an increasing concentration of resources and a decline in the number of competitors; although factors such as growing production value suggest this conclusion, the legal costs of software patents could be a contributing factor.

While the largest developers are acquiring arsenals of defensive patents, the rest of the industry has declined to apply for software patents. One would hope that small innovators and experimental companies would benefit most from software patents, since they are the most vulnerable to having their inventions appropriated without compensation and developed into competing products. However, the costs and difficulties with software patents have led most developers to continue using trade secret protection, and only large companies have opted into software patents.[94] Unfortunately,

[94] *See*, Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("For the most part, the "software industry," as reflected by the above listed companies, really doesn't do much patenting. Only three companies, Apple, Microsoft, and Sun, made the top 50 list of patentees.")

reliance on trade secret cannot protect a developer from infringement suits that, even if groundless, may bankrupt smaller companies with legal fees and settlements.

Worse, software patents may be damaging the industry in several ways. In addition to increasing the costs of software development with no tangible benefit and raising a barrier to entry to exclude new developers, software patents may bring an end to the open-source movement. Open-source projects, in which the developer provides the source code instead of or in addition to the compiled product, have many benefits, such as collaborative and public development, easy customization by users to suit their particular needs, an incentive to write efficient and robust code, and public cooperation in finding and eliminating bugs. However, by providing source code, an open-source developer freely provides closed-source competitors with an easy method of detecting their infringement of a software patent. For this reason, free release of source code may become too dangerous a practice for fear of inviting an infringement threat.

Another unfair practice that has arisen in regard to software patents is known as "landmine patenting."[95] A software innovator publishes a description of his invention in a technical journal, which traditionally implies that readers are free to use it. However, the inventor secretly applies for a patent, quietly pursues a convoluted and protracted course of prosecution while the industry grows to accept and make widespread use of the invention, and then begins demanding license fees for its widespread use.[96] Such

[95] *See*, U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE AND INTELLECTUAL PROPERTY – BACKGROUND PAPER 9 (Global Press Publications, 1990) ("Another problem is the long lag time between patent application and issuance, compared to quick-moving software life cycles. Someone may develop and bring a software package to market, unaware that it will infringe on a patent applied for by another developer, but not yet granted. These are called "landmine patents," and can occur in other areas of technology besides software.")

[96] One example of a "landmine patent" is U.S. Patent No. 4,558,302, issued to Unisys Corp. for the Lempel Ziv Welch (LZW) compression algorithm that forms the basis for Graphics Interchange Format (GIF) image compression. The GIF format is one of the three most widely used methods of storing data for a

"landmines" are undetectable by victims; even if they look for a patent on the described process before using it, the organizational mess of the classification system prevents them from locating it.

Finally, the problems with software problems have created a strong backlash against the patent system. The software patent system has been called a "disaster" and, ironically, "the single greatest threat to innovation in cyberspace."[97] The courts have been faulted for implementing a system "without anybody thinking through the consequences,"[98] the patent office has sustained wide-ranging criticism for its ineffective examination of software patents, and patent practitioners and patentees are accused, in some cases justifiably, of exploiting problems in the patent office and abusing patents. Such criticism may spur Congress or the courts to bring an untimely end to a system of protecting innovation that could prove beneficial if allowed to outlast its growing pains.

V. SOLUTIONS TO PROBLEMS WITH SOFTWARE PATENTS

The problems with software patents largely stem from the youth of the system. Some of the same complaints have been lodged against biotechnology patents, which were also made available to the industry with little warning. An emerging field of science invariably undergoes a bevy of problems adapting into a profitable, efficient industry, but in all cases, these kinks have been eliminated over time; one can expect that

---

picture, and has been in heavy use, especially in conjunction with web pages, for over a decade. However, Unisys recently announced its intention to begin enforcing its patent rights liberally, prompting a public outcry.

[97] James Gleick, "Patently Absurd", *available at* http://www.around.com/patent.html ("'This is a disaster,' says Lawrence Lessig, a Harvard law professor and cyberspace expert. 'This is a major change that occurred without anybody thinking through the consequences. In my view, it is the single greatest threat to innovation in cyberspace, and I'm extremely skeptical that anybody's going to get to it in time.'")

[98] *Id.*

the problems with software patents should do the same. Undoubtedly, the patent office will hire and train more competent examiners; its prior art library for software will become robust, complete, and better organized; and examination procedures will more accurately determine the patentability of software inventions. In the long run, few reasons exist to doubt that patents will eventually benefit the software industry as it has every other technological industry – if it is permitted to outlive its infancy.

As a result, some argue that the best approach to software patents is to do nothing, and this is the recommendation of the U.S. Advisory Commission on Patent Reform.[99] Advocates suggest that the abundance of criticism exists because patentees are discouraged from speaking up in support of software patents, because any such statements could be used against them at trial.[100] It has also been suggested that the punitive effects of software patents result from a necessary and beneficial change in the industry and solves a traditional disincentive to innovation.[101] When one inventor spends research and development costs to create a new, useful application, competitors crank out clones and minor improvements on the idea to steal market share without sharing the costs of development.[102] Software patents that protect the innovator, therefore, may shift

[99] *See*, Software Patent Institute, "The Challenge of Software-Related Patents: Abstracts and Introduction," *available at* http://www.spi.org/primintr.htm ("[T]he US Advisory Commission on Patent Law Reform recommends that the current patent system remain intact but that USPTO and others involved in the patent process obtain better resources on software technologies.")
[100] *See*, Paul Heckel, "Debunking the Software Patent Myths," *Communications of the ACM* June 1992, *available at* http://www.heckel.org/Heckel/ACM%Paper/acmpaper.htm ("[M]ost of the articles on patents in the trade, business and even academic press read by the computer community… have a an [sic] anti-software patent bias. The reason is that for every patent there is one patentholder who is reluctant to speak because the issue is complex and what someone says could e used against them in litigation.")
[101] *See, id.* ("If software patents were more widely respected we would probably have had fewer variations on a theme, and more themes on which to vary. Product development effort seems to have focused on creating many versions of an invention once its value was proven.")
[102] *See, id.* ("Over 250 different spreadsheets and at least four products generally considered to be HyperCard clones were marketed.")

development resources away from building many variations on one idea to the creation of new inventions and applications.

Despite these arguments, *laissez-faire* proponents are in the minority. Theories on how to "fix" software patents are plentiful and sundry. The most sweeping arguments call for the abandonment of software patents altogether or the replacement of the software patent system with a wholly different form of protection. Others propose modifications of the patent examining process or fundamental changes in the scope or duration of software patents. Sill others believe that the impact of software patents can be improved by tactical changes of software developers and patentees relating to software patents, without involving a legislative change that may have unpredictable, adverse effects.

A. RADICAL SUGGESTIONS: ABANDONMENT OF SOFTWARE PATENTS OR *SUI GENERIS* PROTECTION

Many developers call for the complete abandonment of software patents, enumerating the serious problems with software patents as evidence that the system is fundamentally flawed and harmful to software development. Instead, they rely on the traditional forms of protection found in copyright and trade secret. The charge against software patents appears to be led by the open-source movement, the League for Programming Freedom, and smaller developers that find software patents economically unfeasible.

Their primary argument is that software patents are unnecessary to encourage innovation, because vast innovation is one of the traditional hallmarks of the software industry. In fact, innovation is so inextricably intertwined with software development

that the same principle is frequently rediscovered by independent developers – which presents a boon to cooperative development and development of the science of software, but poses a problem with software patents.[103] Others note that no evidence has been found to conclude that software patents do, in fact, encourage innovation; rather, it has been observed that software patents correlate with the size of the software industry, suggesting that software patents are not being used to protect innovation, but to gain leverage over competitors.[104]

Others suggest the development of *sui generis* protection geared more directly toward the nature of the software industry. *Sui generis* acts are a separate form of intellectual property protection carved out of the other kinds of protection for a special purpose, such as the Semiconductor Chip Protection Act.[105] In the late 1970's, semiconductor designers complained that Asian plants were copying their designs; for example, Intel Corp. testified that while current designs cost $10 million to create and test, their chips could be reverse-engineered and copies for as little as $50,000.[106] Because no existing form of protection was powerful enough to prevent this copying, an offshoot of copyright was proposed to allow semiconductor designers to register their products and use the powers of the courts to enforce their rights. The Semiconductor

[103] *See*, League for Programming Freedom, "Software Patents: Is This the Future of Programming?", *Dr. Dobb's Journal* 56 (Nov. 1990) ("There will be little benefit to society from software patents because invention in software was already flourishing before software patents, and inventions were normally published in journals for everyone to use. Invention flourished so strongly, in fact, that the same inventions were often found again and again.")
[104] *See*, Greg Aharonian, "1995 U.S. Software Patent Statistics," *available at* http://lpf.ai.mit.edu/Patents/1995-patents.html ("For the most part, the historical trends of the above software patent statistics correlate with no innovation trends in the software industry. They do correlate with the size of the software industry, which they shouldn't.")
[105] 37 U.S.C. § 902 et seq.
[106] *See generally*, LEMLEY, MENELL, MERGES, & SAMUELSON, SOFTWARE AND INTERNET LAW 396 (Aspen Law & Business 2000).

Chip Protection Act, passed in 1984, created a system of registering semiconductor designs, and has proven beneficial to the semiconductor industry.

The success of the Semiconductor Chip Protection Act has led some critics of software patents to advocate *sui generis* protection for software patents. Even the Office of Technology Assessment has advocated the exploration of alternatives to patents and copyrights for software.[107] However, most interested parties disagree, speculating that any *sui generis* protection will carry even more unforeseen problems than currently plague the patent system and that *sui generis* protection will break with the internationally standardized protection available under the Patent Cooperation Treaty. [108]

One suggested form of *sui generis* protection for software patents is the SDKR protection, modeled after the Semiconductor Chip Protection Act.[109] The goals proposed by the SDKR system over the current systems include: unification of the varied protection offered by copyright and patent; providing a clearer, more consistent set of rules than those implemented by the patent office; duration of protection measured by product life cycles; and relaxed standards of patentability, with a focus on "innovation" rather than "inventiveness."[110] The biggest differences between SDKR and patent protection is that patent examination looks to the "new principle" behind the claimed

[107] *See,* U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 8 (Global Press Publications 1993) ("With respect to software, there may be a point where it becomes preferable to complement or substitute for the existing structures, rather than extend the scope of copyright to fit certain aspects of software – perhaps, cumulatively, at the expense of other types of works.")

[108] *See, id.* at 7 ("[T]he majority of legal experts and firms in the industry takes the position that existing structures like copyright and/or patent are adequate to deal with software, that the case law as a whole is evolving appropriately, and that *sui generis* approaches risk obsolescence as the technology changes and lack an established treaty structure providing international protection [e.g., the Berne Convention provides reciprocal copyright protection in over 75 countries].")

[109] SDKR protection is named for its inventors: Paul Samuelson, Randall Davis, Mitchell Kapor, and Jerome Reichman.

[110] *See,* KENNETH NICHOLS, INVENTING SOFTWARE: THE RISE OF "COMPUTER-RELATED" PATENTS 126 ("In the SDKR, protection is granted to any behavior that meets the standard for innovation. In other words, whatever the computer can be made to do is within the SDKR definition and is potentially protectable.")

invention and classifies patents by this principle, whereas SDKR protection focuses on the novel behavior of the software and attempts to categorize software inventions by their real-world equivalents.[111] One advantage of this focus is that infringement is easier to detect, since protected inventors need not reverse-engineer a competitor's product to show infringement, but rather must show that the two programs exhibit similar behavior.[112] Furthermore, SDKR would avoid the problems faced by the current system, whereby software can protected both by copyright and by patents.[113]

Despite these strengths, several problems with the SDKR proposal mitigate against its adoption. Critics note that the shorter protection term is inappropriate since different kinds of software have variable product life cycles: although many products remain commercially profitable only for a few years and deserve only short-term protection, particularly innovative and useful products, like the RSA encryption algorithm, and products with high development costs and extended viability, such as the *NextStep* operating system, deserve longer terms of protection.[114] The plan grants higher

[111] *See, id.*, at 125 ("The SDKR model is a radical departure from the algorithmic model used by the PTO. Instead of examining how the software operates internally, the SDKR examines what the software does externally, that is, how it behaves. Whereas a patent prevents others from duplicating an operational strategy, the SDKR prevents others from duplicating a behavioral strategy.")

[112] *See, id.*, at 127 ("Because the SDKR focuses primarily on overt software behavior, infringement is more easily detected than it is under patent law.")

[113] *See, id.*, at 106 ("Neither copyright nor patent was designed to fit software. The resulting confusion has made software the first technology to be widely protected by both copyright and patent, which is a problem because the two forms of protection had previously been considered to govern mutually exclusive domains. The nagging suspicion is that, if both copyright and patent seem to fit software, then perhaps neither one really does.")

[114] *See, id.*, at 130 ("The two-to-five-year protection envisioned will be long enough for some products, but too short for others. Games, for example, normally have a short market life, while operating systems may persist for decades. A highly innovative product from a small company may not take off for years. An example is *NextStep*, a Unix-based operating system with an innovative interface and development environment. Over the past decade, *NextStep* has attracted a small but loyal following, competing against companies like Microsoft and IBM on the Intel-based network platform. A successful operating system, like *NextStep* (or Unix, *Windows*, or the MacOS), can justify high development costs by remaining viable for many years. Under the SDKR, *NextStep*'s protection would have expired long ago.")

An additional argument along these same lines is that although a software patent is granted for 20 years, which far exceeds the viability of most software inventions, this doesn't harm the industry. After all,

priority to commercial software developers who bring a product to market than to researchers who develop software inventions in a laboratory environment; the latter group must continue to rely on trade secret or another form of protection.[115] It is also worth noting that many useful inventions, such as the popular compression utility PKZIP, are created by one independent developer or small company that lacks the resources to bring a full-fledged product to market, but obtains a patent and licenses the technology to development firms; but they, too, would be excluded from SDKR protection. Finally, the behavioral focus provides no protection for inventions that exhibit no visible "behavior," so many valuable inventions, such as the RSA encryption algorithm, could not take advantage of the protection.

B. SUBSTANTIAL MODIFICATIONS TO SOFTWARE PATENT PROSECUTION OR TO SOFTWARE PATENTS

A second group of proponents for software patent reform argue not for replacement of software patents, but for legislative changes. Their suggestions fall into two categories: proposals for changes in software patent prosecution and changes to the scope and duration of software patents.

---

if an invention loses its market viability before the patent term expires, it has either been fully exploited or outpaced by other technology; therefore, it does not matter whether protection expires immediately after viability ends or much later – the invention has no value left in either case. A counterargument to this theory is that one of the reasons for implementing a patent system is to secure valuable inventions for public use, and the public obtains no benefit by earning the right to use an invention long after it has become worthless. However, this counterargument does not seriously impact the patent system or any other proposed form of protection, including the SDKR, because none of these methods of protection suggest a revocation of protective rights before the invention becomes commercially useless (even if that point in time were reliably measurable.)

[115] *See, id.* at 130 ("Because the SDKR would only give protection to products on the market, any discovery made in the laboratory would have to be maintained as a trade secret or turned into a product simply to gain protection.")

Suggestions of reform for patent prosecution take many forms. Some advocate the adoption of an adversarial patent prosecution system, such that patent applications are open to the public and their merit can be challenged before the patent issues. Such systems have been adopted in other countries and provide a more cost-effective means of challenging patent applications prior to allowance, as well as a stronger presumption of validity of issued patents. However, this system presents its own problems, and the change would require a radical shift in examining procedures, with the examiner taking a more neutral approach in the examination process.

A similar suggestion is the creation of an adversarial reexamination proceeding. After a software patent issues, any member of the public can request a reexamination of the patent in light of new arguments against patentability or prior art that could limit or invalidate the patent.[116] However, the currently established procedure allows the requesting party to provide this initial communication and submission of prior art, and then ends its involvement in the proceedings; if the request for reexamination is granted, the reexamination occurs between a patent examiner and the patentee, to the exclusion of the requester. The process does not allow the responsive debate that accompanies an adversarial proceeding.

Some practitioners have expressed that this *ex parte* process is more akin to a reconciliation of an issued patent with prior art than an actual reexamination of the patent.[117] Accordingly, the patent office is currently developing a new *inter partes*

---

[116] *See*, 35 USC § 301 ("Any person ... may file a request for Reexamination of any claim on the basis of any prior art cited."); MANUAL OF PATENT EXAMINING PROCEDURE § 2209 *et seq*.

[117] *See*, comments of Bruce A. Lehman, former Commissioner of Patents and Trademarks, before Congress, speaking about reexamination reform, *available at* http://www.house.gov/judiciary/419.htm:

> One particularly striking comment is the often-heard advice given to a third party who knows of pertinent prior art that could invalidate a patent. The advice is that the third party not use patent reexamination. Instead, third parties are advised to reserve the prior art for later use in either a

reexamination process that is closer to an adversarial proceeding and may have several advantages.[118] An adversarial reexamination may cost much less than litigation and provide a convenient, cost-effective process for vindicating the rights of parties whose interests have been hindered by an unduly broad patent. An adversarial system would allow such parties to provide their full arguments for restriction or invalidity and reply to responses of the patentee; this affords the requesting party a fairer opportunity to press its case against the patent than the current procedures.[119] And on a broader scale, the resulting elimination of invalid software patents would clean up the body of software patents and restore industry confidence in the validity of patents that survive such proceedings.

One novel suggestion is the revival of a long-abandoned practice in patent prosecution. The early patent acts required an applicant to submit a working model of the invention during prosecution.[120] These models demonstrated that the applicant had

---

> negotiation with the patent owner, or an infringement action in Federal court. Individuals advocating this stance point to the absence of effective third-party Participation in the reexamination proceeding coupled with a perception of "enhanced" validity that a reexamined patent enjoys, especially before a jury. Comments like these suggest to me that as far as third parties are concerned, there is a significant lack of public confidence in the current reexamination system as an effective means for challenging the validity of patents.

[118] *See*, Reexamination Reform Act of 1995; "Patent Litigation Reduction Act," Title V of the American Inventors' Protection Act.

[119] *See*, Patricia D. Granados, Esq., "Reexamination Reform Act of 1995," *available at* http://www2.arj.net/foley/reexam.html ("Under the proposed law, a third-party requester would have one opportunity to rebut every argument the patent owner makes and can even participate in interviews. A third-party requester also would be permitted to appeal an adverse decision to the PTO's Board of Patent Appeals and Interferences ('Board') and to the U.S. Court of Appeals for the Federal Circuit ('Federal Circuit') or be a party to an appeal filed by the patent owner. As such, it provides a more adversarial forum for determining the validity of a patent and, arguably, is a fairer substitute for litigation.")

[120] *See*, Patent Act of 1790 § 2 ("[T]he grantee or grantees of each patent shall, at the time of granting the same, deliver to the Secretary of State a specification in writing, containing a description, accompanied with drafts or models, and explanations and models (if the nature of the invention or discovery will admit of a model) of the thing or things, by him or them invented or discovered…."); Patent Act of 1793 § 3 ("[An] inventor shall, moreover, deliver a model of his machine, provided, the secretary shall deem such model to be necessary."); Patent Act of 1836 § 6 ("[An applicant] shall moreover furnish a model of his invention, in all cases which admit of a representation by model, of a convenient size to exhibit advantageously its several parts.")

successfully reduced the claimed invention to practice and ensured that the disclosure was enabling, and the models were kept on display in the patent office to allow the public to examine them. The suggestion has been raised that the requirement be applied to software patents, such that applicants must submit source code with their application.[121] Currently, because no such requirement exists and patent examination is perfunctory, software patent applicants are free to withhold source code and describe their software invention in such vague terms as to raise questions of breadth and enablement. This complicates both a comparison of the application with prior art references to determine obviousness and anticipation and a comparison of a patent issuing on the application with *subsequent* applications and the products of accused infringers. On the other hand, a requirement that software applications include source code would provide the examiner with a clearer basis for these comparisons; act as evidence that the applicant has, in fact, reduced the claimed invention to practice; and provide a more useful disclosure of the invention to the public.

Some software patent advocates urge the patent office to raise the threshold of inventiveness above the "novel and non-obvious" standard.[122] Because software is such a

[121] *See*, Jason V. Morgan, "Chaining Open Source Software: The Case Against Software Patents," *available at* http://lpf.ai.mit.edu/Patetns/chaining-oss.html ("Requiring software patents to include reference source code would be similar to the old requirement that patents include a physical model. IT could be argued that since patents no longer include this requirement that software should not have a similar requirement, but such a requirement would be a good way to reform a system that "disproportionately rewards ideas rather than their application.")

[122] The Patent Act of 1836 § 1 required the newly-established patent office to ensure that patented inventions were "new." This standard was employed until the late 1800's, when the federal courts began tightening the restrictions on novelty such that inventions were patentable only if they required a "flash of creative genius" to imagine and design in light of the prior art. Indeed, the courts' bias against patent applicants continued to rise throughout the first half of the 1900's, culminating with a well-known observation by Mr. Justice Jackson of the Supreme Court that the "the only patent that is valid is one which this Court has not been able to get its hands on." (Jungersen v. Ostby & Barton Co., 335 U.S. 560, 572 (1949).) Congress reversed this trend with the Patent Act of 1952, which instituted the more lenient "non-obviousness" standard that is currently employed by the patent office and the federal courts. The

dynamic field, demanding much more innovation in the development of a particular algorithm than is typically required for inventions in other fields, perhaps the examiners should not only grant patents for those inventions that reveal a "flash of creative genius."[123] While implementing different levels of examination for different kinds of inventions may prove unworkable, this change would provide examiners more leeway in rejecting applications of little merit and raise the value of the invention claimed in the average software patent.

A similar suggestion is to increase the specificity of software patents. This could be achieved either by requiring a heightened level of disclosure and more narrowly tailored claims in the patent application, or by restricting the breadth of claim construction and the range of the doctrine of equivalents that the courts are willing to apply. However, one might wonder whether the concomitant restriction of the rights of software developers, already beleaguered by the high cost and long delay in obtaining a patent and the difficulty of enforcing software patent rights, may further withdraw from the patent system in favor of alternative forms of protection.

One of the more frequent suggestions is a shortening of the term of a software patent.[124] A patent is a delicate balancing of protecting the short-term, but exclusive, rights of the inventor against protecting the long-term interest in the public disclosure of useful inventions that eventually pass into the public domain. However, a 20-year term for software patents may defeat the latter purpose: by the time the patent lapses and the

---

suggestions of software patent advocates discussed here amount to a return to the "flash of creative genius" standard, such that software inventions must demonstrate a *significant* leap of innovation over the prior art.

[123] Jungersen v. Ostby & Barton Co., 335 U.S. 560, 572 (1949).

[124] *See*, KENNETH NICHOLS, INVENTING SOFTWARE: THE RISE OF "COMPUTER-RELATED" PATENTS 104 ("Software is different from mechanical and chemical inventions because of its brief market life span – twenty years is much too long to protect software.")

invention passes into the public domain, the invention may have passed into obsolescence, and the public will not benefit from the disclosure.[125] The dynamic nature of the software field and the short lifespan of software products and inventions have led some to conclude that a 20-year monopoly provides far too much coverage; one economist has argued that "technology moves now with a speed once undreamed of – its swift march dictates a shortening in the life of a patent."[126] Even Jeff Bezos, CEO of Amazon.com and advocate of the software patent system, argues in favor of a three- to five-year term for software patents.[127] On the other hand, reasons for maintaining the current 20-year term include the fact that the 20-year term was established recently to match the term of patents offered by foreign patent offices, so that altering the patent term for any class of inventions may jeopardize the reciprocal patent treaties[128] and threaten foreign enforcement of U.S. patents;[129] and the fact that some software is so novel or durable that it may well outlast a five-year patent term.[130]

Another suggestion for change is to create a "safe harbor" for developers of free, open-source products by limiting or negating their potential liability for infringement.

---

[125] Bruce Perens, "Bruce Perens on Patents and Politics," *available at* http://sendmail.net?feel=interviewperenspatents ("For each industry, you have to ask, 'How long is it before a new idea becomes obsolete and useless?' In mechanical engineering, that may be a person's lifetime. In computer science, it may be five years or seven. Look what we were doing five years ago: people have *thrown those computers out.* So the question is: Is 20 years too long for a computer science patent?")

[126] U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 196 (Global Press Publications, 1993).

[127] *See*, Philip E. Ross, "Patently Absurd," *Forbes* May 29, 2000, *reproduced at* http://www.forbes.com/forbesglobal/00/0529/0311090a.htm ("Another reform is offered by none other than Jeffrey Bezos, the Amazon founder and master of the all-embracing software claim. He suggests limiting software patents to three to five years, enough time to grant a head start without stifling competition.")

[128] *See, e.g.*, Patent Cooperation Treaty; Uruguay Round Agreements Act.

[129] This may not be so great a consideration because the European Patent Office has taken a firm stance against software patents, in part because of the numerous problems that the U.S. Patent and Trademark Office has experienced with software patents. One wonders whether limiting the duration of software patents would actually have an impact on reciprocity, since foreign patent offices have little respect even for software patents that carry the standard 20-year term.

[130] One example of such software is the RSA encryption algorithm, which remained profitable and useful throughout its patent protection and is still in heavy use today.

Commentators have observed the threat to open-source software posed by the software patent system,[131] but an open-source safe harbor may insulate the industry from the threat of infringement that such developers face. Because open-source software is commonly free to the public, it is less attractive to investors; such developers have fewer development funds to spend on patent infringement prevention and defense, and therefore may be entitled to a limited or absolute exemption from software infringement suits. Furthermore, the public is benefited by being granted access to the source code in order to tailor software for their own purposes, but free access to source code also serves to assist patentees with detecting infringement, rendering open-source developers even more vulnerable to patent infringement suits. Therefore, the public interest may require a "safe harbor" provision limiting or barring software patent enforcement against open-source developers. A "safe harbor" provision would allow open-source developers to operate without fear of infringement suits that they cannot predict, avoid, or contest at trial. On the other hand, open-source developers would have every incentive to copy inventions wholesale from the public disclosure that accompanies every patent, thus depriving software patentees of a substantial measure of protection over their claimed invention.

C. TACTICAL SUGGESTIONS: NEW STRATEGIES FOR CHALLENGING SOFTWARE PATENTS

[131] *See*, Jason V. Morgan, "Chaining Open Source Software: The Case Against Software Patents," *available at* http://lpf.ai.mit.edu/Patents/chaining-oss.html ("Software patents have hindered the proliferation of open source software in the past and they could easily damage the future of open source software and the rest of the software industry.")

The initial establishment of the software patent system by edict of the federal courts over the objections of the patent office[132] and other agencies[133] has led to the wealth of problems already discussed. Some have observed that the federal courts' hasty retreat on limitations of software patents has led to unforeseeable consequences,[134] and therefore argue against further formal changes in the software patent system that may further destabilize the software industry. Instead, such proponents advocate tactical changes by the software industry with regard to its software patent practices.

One common suggestion aimed to remedy a number of problems is a cooperative effort of software developers to develop a public prior art database.[135] This would allow applicants to search the prior art to craft their claims more carefully, patent examiners to search for prior art in an industry-maintained and industry-referenced database, and software developers to find prior art and avoid or defend against infringement claims with more predictability and accuracy. The strongest effort to create such a system is the Software Patent Institute ("SPI"),[136] sponsored by many large software developers and frequent software patent applicants, including Microsoft, Apple, and IBM. SPI accepts

---

[132] *See,* G. GERVAISE DAVIS III, ESQ., SOFTWARE PROTECTION: PRACTICAL AND LEGAL STEPS TO PROTECT AND MARKET COMPUTER PROGRAMS 155(Van Nostrand Reinhold Co. Inc. 1985) ("The Patent Office and the CCPA, as the then principal court for patent appeals, engaged in a tug-of-war for a few years over software patentability. The Patent Office's position was that it did not have the facilities to deal with software patents, so it did not want software to be patentable.")

[133] *See,* U.S. CONGRESS, OFFICE OF TECHNOLOGY ASSESSMENT, COMPUTER SOFTWARE: COPYRIGHTS, PATENTS, AND THE CHALLENGE OF TECHNOLOGICAL CHANGE 12 (Global Press Publications 1993) ("[T] he PTO has grappled with several institutional problems, including issues such as: examiner training and turnover, length of pendency periods (from filing to issuance) for patent applications, a backlog of applications, and the quality and extent of the prior art database…. In OTA's view, these problems are serious in that they may affect the quality of the patents issued and create additional burdens for software developers and users.")

[134] *See,* James Gleick, "Patently Absurd", *available at* http://www.around.com/patent.html ("This is a major change that occurred without anybody thinking through the consequences.")

[135] *See,* Wayne M. Kennard, "Obtaining and Litigating Software Patents and Protecting Software on the Internet," *reproduced in* PETER BROWN, 16TH ANNUAL INSTITUTE ON COMPUTER LAW 291 (1996) ("The search for non-patent prior art is more difficult. In part, this is because there is no central depository which can be searched. … A relatively new search facility, the Software Patent Institute ("SPI"), has been established to provide a database geared for searching for prior art.")

[136] Available at http://www.spi.org .

submissions by the public and provides free access to "the 'folklore' of the industry"[137] – those instances of prior art that are not patented or discussed in depth in recent technical journals, which is a current trouble spot in software patents because this information is difficult to find in print. However, the incentive to contribute in order to avoid future patent infringement suits is balanced against the fact that publication teaches useful techniques to competitors that could otherwise be protected by trade secret. And, while submission to SPI does not prevent a contributor to file a patent application on the published technique (if filed within the relevant deadlines[138]), this raises the spectre of "landmine patenting."

Another common method of avoiding infringement suits is defensive publication. Publishing a new software invention in a technical journal, or even through the Defensive Disclosure System maintained by the Software Patent Institute, prevents others from later filing for a patent on the technique because the invention is no longer "novel" as of the date of the publication.[139] Therefore, a software developer may use the invention with less fear of subsequent infringement claims or the expense and uncertainty of software patent prosecution.[140] Defensive publication is especially effective when the substantial patentability of the invention is questionable, but the weaknesses of software patent examination might allow a competitor to develop the invention independently and obtain

[137] "Introduction to the SPI Database," available at http://www.spi.org/freedbasics.html .
[138] *See*, 35 USC § 102(a) (an invention is unpatentable if "known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for patent"), § 102(b) (an invention is unpatentable if "described in a printed publication in this or a foreign country… more than one year prior to the date of the application for patent in the United States.")
[139] *See*, *id*.
[140] *See*, Software Patent Institute, "The Challenge of Software-Related Patents: Primary Uses," *available at* http://www.spi.org/primuses.htm ("[O]ne can prevent others from patenting software technology that is vital to his or her business, without the expense of obtaining a patent, by publishing the software technology. SPI will publish this information in a database that is accessible to USPTO, members of SPI and the general public.")

a patent on the invention that could later be used to threaten other developers. However, the benefits of defensive publication are offset by several hazards. Defensive publication does not afford the same protection as a patent, and cannot be used as leverage in a cross-licensing scenario.[141] Defensive publication also surrenders trade secret protection and teaches the invention to competitors, allowing them to use the invention without the expense of developing them independently.[142] Finally, because publications only foreclose patentability of applications filed after the date that publication issues, a competitor who has independently developed the invention and applied for a patent before the defensive publication date can use the publication as strong evidence of infringement once the patent issues.

One proposal for avoiding patent infringement suits is a mutual defense policy, whereby all members of the policy agree to use all of their software patents for defensive purposes only.[143] Membership is easy to obtain, but members expressly cross-license all current patents to all other members and agree not to file software patent infringement suits against any other developer. The stated goal of this proposal is to swell membership in the policy to the point where "most software developers will find membership

[141] *See*, KENNETH NICHOLS, INVENTING SOFTWARE: THE RISE OF "COMPUTER-RELATED" PATENTS 104 ("Even though there are other defensive strategies (for example, the Software Patent Institute maintains the Defensive Disclosure Service, whose purpose is to enable developers to avoid the need for defensive patenting [Syrowik et al. 1994]), such strategies are not as likely to discourage infringement suits as a patent covering the accused protect.")

[142] *See*, Software Patent Institute, "The Challenge of Software-Related Patents: Disclosing Software Inventions to the SPI," *available at* http://www.spi.org/primdisc.htm ("Just like any other form of publication, publication by SPI will destroy any trade secret rights one might have in the published information. … Anyone who submits information to SPI for publication risks giving his or her competitors an understanding of his or her business.")

[143] *See*, League for Programming Freedom, "Mutual Defense Against Software Patents," *available at* http://lpf.ai.mit.edu/Patents/mutual-def.html ("Software developers can protect themselves more effectively from patents by explicitly adopting a nonaggression or mutual defense policy. This means promising in a binding fashion to use their patents only to protect themselves and others from patents.")

essential."[144] However, it is apparent that this suggestion functions as a general retreat from software patents. Were the policy to become so large that most developers were involved, very few patents would be enforceable against most of the software industry – thereby rendering moot the purpose of obtaining any software patent. Much like the defensive patenting scheme, even patents meritorious inventions, such as the RSA algorithm, will remain publicly divulged but unenforceable against competitors.[145] Furthermore, this system would create an incentive for developers to use others' patented (and publicly disclosed and taught) software inventions for free rather than investing in research and development of new software inventions; and those developers who continue to innovate will have stronger incentive to rely on trade secret than to apply for a software patent that is costly, time-consuming, and unenforceable and requires disclosure of the invention. Therefore, one wonders what benefit a mutual defense policy presents over a complete abandonment of the software patent system.

Finally, the current reexamination process currently employed by the patent office may be used to protect innovation without the need for radical change or legislative revision of the patent system. As discussed above, any member of the public may file a request for reexamination along with arguments for the limitation or invalidity of the patent and relevant prior art references that have not already been considered during prosecution or former reexamination proceedings.[146] Although current reexamination proceedings are largely *ex parte*, with the reexamining patent examiner assuming the

[144] *See, id.* ("This organization would strive to be universal, so it would be well publicized and easy to join. Once the organization becomes large enough, most software developers will find membership essential, which will assure continued success.")
[145] *See, id.* ("Any software developer that tries to use patents for attack will immediately become fair game for all.")
[146] *See*, MANUAL OF PATENT EXAMINING PROCEDURE § 2209 *et seq.*

interests of the requesting party and the public, and not truly adversarial, reexamination does afford an opportunity for invalid patents to be stricken at a fraction of the cost of litigation. Furthermore, a reexamination request may be filed at the requesting party's convenience and is not constrained by the time limits that normally accompany infringement suits, and although adverse findings by the examiner on reexamination carry *some* weight, they do not cause *res judicata* restrictions against raising the same arguments for invalidation before a federal judge during an infringement claim.

The benefits of reexamination could be more strongly realized if many members of the industry and academia cooperated to formulate strong reexamination requests. The traditional problem of finding prior art in light of widespread reliance on trade secret protection could be overcome if developers had an incentive (namely, the invalidation of a potentially threatening patent) to provide prior art examples from past or current projects. Moreover, such organized efforts distribute the burden of supplying prior art examples to all developers, whereas infringement suits place the entire burden on the accused infringer(s), who may not be able to mount a sufficient defense against a questionable patent. It is anticipated that an organized effort to call for a reexamination would better represent the consensus of the entire industry as to which patents are valuable and which are merely new (or even old) embodiments of the prior art.

This system has many advantages. An organization funded by the software industry and charged with organizing efforts for reexamination requests could be organized without the need for legislative change that may entail unpredictable results. Such a system would transfer some of the responsibility of policing the body of software patents on competing developers, rather than placing critical decisions of patentability

entirely within the realm of examiners who are not experts in the field and cannot find prior art easily. By invalidating questionable patents through reexamination procedures, the software industry could place more significance on those software patents that remain, and software development would become a safer process without questionable software patents clogging the software patent system. Furthermore, if patents that are questionably meritorious but expensive to obtain begin to be struck down by less expensive calls for reexamination, patent applicants would have an incentive to limit patent applications only to *clearly* meritorious inventions. Even the increased scrutiny given to issuing patents may discourage software developers from applying for patents on questionably valid claims.

In summary, industry-wide participation in a process of organizing requests for reexamination of meritless patents could well be strong medicine for negating invalid patents and curing many of the ills of the software patent system.

VI. CONCLUSION

The current system of software patents faces many problems and challenges for the growing software industry. Some problems may require radical change to overcome; some demand structural changes that could benefit the industry; others merely need time to resolve. Nevertheless, debate on the proper solution ranges from complete abandonment or replacement of the software patent system to more subtle changes in procedure or industry tactics. In fact, the only clear aspect of the software patent problem is that no consensus exists as to how to improve the system – or if improvement is even necessary.

In the long run, however, there is little reason to believe that the software industry is so different from any other industry that the rewards of a patent system, which have proven strongly beneficial for every other industry, do not apply to software and cannot spur innovation. A careful, balanced review by Congress and the patent office is needed to determine what changes are essential to help the software patent system outlive its youth and to best integrate patents with the software industry.