NYLS Law Review

Volume 55
Issue 1 *D Is for Digitize*

Article 9

January 2011

Software Wars: The Patent Menace

Andrew Nieh
New York Law School Class of 2010

Follow this and additional works at: https://digitalcommons.nyls.edu/nyls_law_review

Part of the Communications Law Commons, Computer Law Commons, Intellectual Property Law Commons, Law and Society Commons, and the Legal Remedies Commons

Recommended Citation
Andrew Nieh, *Software Wars: The Patent Menace*, 55 N.Y.L. SCH. L. REV. 295 (2010-2011).

This Note is brought to you for free and open access by DigitalCommons@NYLS. It has been accepted for inclusion in NYLS Law Review by an authorized editor of DigitalCommons@NYLS.

VOLUME 55 | 2010/11

ANDREW NIEH

Software Wars: The Patent Menace

ABOUT THE AUTHOR: Andrew Nieh received his J.D. from New York Law School in May of 2010.

If people had understood how patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be at a complete standstill today.[1]

I. INTRODUCTION

Computers are an essential part of our everyday lives—it is hard to imagine a single day when we are not sitting in front of a computer. Software is the driving force behind these machines, providing the instructions that are necessary to make our computers[2] and the applications on which we depend run.[3] Yet our society is still debating whether the legal protections that apply to software are appropriate, or whether a new paradigm is necessary. Deciding what legal protection should be afforded to software, however, involves competing policy considerations.[4] Ultimately, the laws protecting software need to properly balance the economic incentives that will encourage people to develop software with the competitive considerations that will allow those in the software industry[5] to continue innovating.

Currently, copyright and patents are the predominant legal tools used to protect the intellectual property rights of software developers in the United States.[6] The Progress Clause of the U.S. Constitution authorizes Congress to enact laws "[t]o 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."[7]

1. Memorandum from Bill Gates, Pres. & Chairman, Microsoft, Inc. on Challenges and Strategy to the Executive Staff (May 16, 1991), *available at*, http://web.archive.org/web/20010218085558/http://bralyn.net/etext/literature/bill.gates/challenges-strategy.txt.

2. *See* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1243 (3d Cir. 1993). Software is also integral to running our mobile devices, such as smartphones, iPods, and the GPS systems in our cars, among many other things.

3. *See id.*

4. *See generally infra* note 15.

5. Defining the exact scope of the "software industry" is difficult because many organizations that are not considered software companies are in some respect engaged in software development, whether through the creation of business tools, programs to run electronic devices, or internet websites, for example. Studies often consider organizations which are only engaged in software publishing as the real software industry, and these same studies have found that patents have little or no negative effect on these specific organizations. *See* James Bessen & Michael J. Meurer, Patent Failure: How Judges, Bureaucrats, and Lawyers Put Innovators at Risk 189–90 (2008) (discussing these studies, but noting that "the software-publishing industry only obtains 5 percent of all software patents granted; most are obtained by firms in electronics, telecommunications, and computer industries"). This note uses "software industry" broadly to include any individual or organization engaged in developing or producing software. Accordingly, for the purposes of this note, progress and innovation in the software industry refers to the ability of any software developer to engage in the creation or improvement of software technology in a fair and competitive market.

6. Trade secret law also offers software developers legal protection. *See generally* Julie E. Cohen et al., Copyright in a Global Information Economy 266 (2d ed. 2006) (describing the scope of protection of trade secrets for software).

7. U.S. Const. art I, § 8, cl. 8. This clause is also referred to as the Intellectual Property Clause or the Copyright and Patent Clause.

There is much dispute, however, about whether these rights actually "promote . . . Progress"[8] in the software industry.[9]

Recently, the validity of software patents under the Patent Act of 1952 has been subjected to legal scrutiny. While software is explicitly protected under the Copyright Act,[10] it is protected under patent law only through judicial interpretation of the Patent Act.[11] In 2008, the U.S. Court of Appeals for the Federal Circuit decided *In re Bilski*, a controversial case that essentially limited the scope of patent protection for business methods under the Patent Act.[12] Although the Federal Circuit refused "to

8. *Id.*

9. Software patents are particularly controversial and generate a vast amount of argument among economists, legal scholars, and business people as to whether these patents hinder software innovation. *See, e.g.*, F. Scott Kieff et al., Principles of Patent Law 845 (4th ed. 2008) ("Some economists have cast doubt on the need for software patents, or worse, assert they are harmful to software innovation." (citing James Bessen & Robert M. Hunt, *An Empirical Look at Software Patents* (Fed. Bank of Phila., Working Papers 03-17, 2003))); William M. Landes & Richard A. Posner, The Economic Structure of Intellectual Property Law 326 (2003) ("There is also evidence that the patenting of computer software actually retards innovation . . . ."); Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 Colum. L. Rev. 2308, 2343–47 (1994) (explaining how patent law is "ill-suited to protecting software innovation"); James Bessen & Eric Maskin, *Sequential Innovation, Patents, and Imitation* 2 (Dep't. of Econ., Working Paper No. 00-01, 2000) ("For industries like software or computers, there is actually good reason to believe that imitation *promotes* innovation and that strong patents (long patents of broad scope) *inhibit* it.").

10. 17 U.S.C. § 117 (2006) (providing for certain limitations on the scope of exclusive rights granted to copyright holders of computer programs); 17 U.S.C. § 101 (2006) (defining "computer program" as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result"). As for its subject matter classification, software is considered a "literary work" under § 102 of the Copyright Act. *See* 17 U.S.C. § 102(a)(1) (2006). See also Atari Games Corp. v. Nintendo of Am., Inc., 975 F.2d 832, 838 n.2 (3d Cir. 1992). The *Nintendo* court stated that:

> [t]he statutory definition of "literary works" embraces computer programs: "Literary works" are works, other than audiovisual works, expressed in words, numbers, or other verbal or numerical symbols or indicia, regardless of the nature of the material objects, such as books, periodicals, manuscripts, phonorecords, film, tapes, disks, or cards, in which they are embodied. As works "expressed in words, numbers, or other verbal or numerical symbols or indicia," computer programs fall within the terms of the 1976 Act. The House Report for the 1976 Act explicitly includes computer programs within "literary works": The term "literary works" does not connote any criterion of literary merit or qualitative value: it includes . . . computer data bases, and computer programs to the extent that they incorporate authorship in the programmer's expression of original ideas, as distinguished from the ideas themselves. As literary works, copyright protection extends to computer programs.

Id. (citing H.R. Rep. No. 94-1476, at 47 (1976), *reprinted in* 1976 U.S.C.C.A.N. 5659, 5666); Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1249 (3d Cir. 1993).

11. *See, e.g.*, Diamond v. Diehr, 450 U.S. 175, 177, 184, 187 (1981) (holding that "a process for curing synthetic rubber which includes in several of its steps the use of a mathematical formula and a programmed digital computer is patentable subject matter" after engaging in statutory construction of the term "process" in § 101 of the Patent Act and further holding that "a claim drawn to subject matter otherwise statutory does not become nonstatutory simply because it uses a mathematical formula, computer program, or digital computer").

12. 545 F.3d 943 (Fed. Cir. 2008), *aff'd sub nom.* Bilski v. Kappos, 130 S. Ct. 3218 (2010). One of the threshold prerequisites to receiving a patent is having an invention or process that is directed towards

adopt a broad exclusion over software" under the Patent Act,[13] its ruling still had a detrimental effect on patent applications for software-related claims.[14] Further, the case renewed the policy debate on whether software merits patent protection.[15]

On June 28, 2010, the Supreme Court decided *Bilski v. Kappos*, which rejected the Federal Circuit's machine-or-transformation test as the sole test for determining whether a claim is a process for statutory subject matter purposes, holding that the test was merely a "useful and important tool" in finding a process claim.[16] The Supreme Court, however, did not reach the issue of whether software was patentable subject matter, leaving the validity of software patents up in the air.[17]

---

one of the four classes of statutory subject matter. *See* 35 U.S.C. § 101 (2010) ("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). The Federal Circuit upheld the Board of Patent Appeals and Inferences' rejection of "a method of hedging risk in the field of commodities trading" claim "as not directed to patent-eligible subject matter under 35 U.S.C. § 101." *In re Bilski*, 545 F.3d at 949. In doing so, the court enunciated a new test for statutory subject matter: the "machine-or-transformation test." *Id.* at 954 n.7. "The machine-or-transformation test is a two-branched inquiry; an applicant may show that a process claim satisfies § 101 either by showing that his claim is tied to a particular machine, or by showing that his claim transforms an article." *Id.* at 961. *In re Bilski* was appealed to the U.S. Supreme Court and was affirmed. *See* Bilski v. Kappos, 130 S. Ct. 3218 (2010). While the main issue on appeal before the Supreme Court was whether Bilski's business method was properly rejected under the new statutory subject matter standard, amici curiae had petitioned the court to also consider the subject matter eligibility of software; but, the Court declined to consider this issue. *Bilski*, 130 S. Ct. at 3228.

13. *In re Bilski*, 545 F.3d at 960 n.23.

14. *See, e.g.*, *Ex parte* Becker, No. 2008-2065, 2009 WL 191977 (B.P.A.I. Jan. 26, 2009) (rejecting a claim for a "'method for creating a hierarchically structured automation object and embedding said automation object into an engineering system'" as not directed towards statutory subject matter under *Bilski*'s machine-or-transformation test); *Ex parte* Barnes, No. 2007-4114, 2009 WL 164074 (B.P.A.I. Jan. 22, 2009) (rejecting a claim for a "method for indentifying faults in a seismic data volume" as "directed to non-statutory subject matter" under *Bilski*'s machine-or-transformation test); *Ex parte* Gutta, No. 2008-3000, 2009 WL 112393 (B.P.A.I. Jan. 15, 2009) (rejecting a claim for a "computerized method performed by a data processor for recommending one or more available items to a target user" (i.e., a computer program) as directed towards non-statutory subject matter under *Bilski*'s machine-or-transformation test); *see also Ex parte* Cornea-Hasegan, No. 2008-4742, 2009 WL 86725 (B.P.A.I. Jan. 13, 2009) (rejecting "a method for predicting results of floating point mathematical operations and calculating the results . . . using software rather than hardware . . . when the results are tiny" as non-statutory subject matter).

15. *See, e.g.*, Brief of Amicus Curiae Red Hat, Inc. in Support of Appellee, *In re* Bilski, 545 F.3d 943 (Fed. Cir. 2008) (No. 2007-1130) (arguing that software should *not* be patented) [hereinafter Red Hat Amicus Curiae Brief]; Brief for the Business Software Alliance as Amicus Curiae Supporting Neither Party and Supporting Affirmance, *In re Bilski*, 545 F.3d 943 (Fed. Cir. 2008) (No. 2007-1130) (arguing that software should be patented); see *also In re Bilski*, 545 F.3d at 1010 (Mayer, J., dissenting) (finding that one of the "thorniest issues in the patentability thicket" is "the extent to which computer software and computer-implemented processes constitute statutory subject matter").

16. *Bilski*, 130 S. Ct. at 3227.

17. *See id.* at 3228 ("It is important to emphasize that the Court today is not commenting on the patentability of any particular invention, let alone holding that any of the . . . technologies from the Information Age should or should not receive patent protection."); *see also* David Worthington, *Supreme Court Strikes Down Bilski Patent Claim*, Software Dev. Times, June 28, 2010, http://www.sdtimes.com/link/34447

Part II of this note discusses the history of software protection under the Copyright Act and Patent Act. It then gives an overview of the predominant theories driving patent protection for software. Part III examines how patents impact the software industry, specifically focusing on their effects on research and development and the resulting increase in litigation. Part IV argues that the problems with software patents are largely the result of an industry and technology which are incompatible with a normative patent system. This incompatibility leads to the issuance of patents which ultimately restrict software development through decreased research and development investments and an increased potential for software developers to face patent infringement lawsuits.

Part V proposes that software should be ineligible statutory subject matter under the Patent Act because patents do not promote progress in the software industry. In hearing the *Bilski* arguments on appeal from the Federal Circuit in its October Term, 2009, the Supreme Court had an opportunity to change the legal and economic landscape by deciding whether software claims are non-statutory subject matter under the Patent Act. However, the Court declined to address the issue, and, thus, the patent protection afforded to software claims remains an open question, which will undoubtedly come before the Court in the future. This note proposes that the Court, when it eventually faces this specific issue, should adopt a per se exception barring all software from patent protection under the rationale that patents do not protect abstract ideas or algorithms—a fundamental patent law principle.

Such a broad holding would not be as drastic as it might seem. The software industry would not suffer any undue hardships from such a change in the legal regime, but may in fact benefit from it. The software industry was thriving before patent protection dominated the industry, and, once software patents became more prevalent, evidence shows that such protection did not achieve the progress that proponents believed it would. Most importantly, the other existing modes of intellectual property protection are more compatible with software and adequately promote progress in the software industry, while providing ample protection for software developers. These rights are also more readily adaptable to the new, important incentive structures that are beginning to emerge in the software industry. One such example is the way copyright law is used by the open source movement[18] as

(quoting Professor James Grimmelmann as saying: "There will be continued uncertainty and confusion around the validity of existing software patents [after *Bilski*] . . . .").

18. The open source movement is a community-based initiative where developers and programmers agree to license their software code royalty-free to the general public, provided that certain conditions are met. Examples of these licenses include the "GNU's Not Unix" (GNU) General Public License, version 2, and the Artistic License. *See, e.g.*, *GNU General Public License Version 2*, GNU Operating Sys., http://www.gnu.org/licenses/gpl-2.0.html (last visited Oct. 29, 2010); The Perl Found., http://www.perlfoundation.org/artistic_license_2_0 (last visited Oct. 29, 2010). These licenses are based on the exclusive rights granted to copyright holders of the software and typically allow the licensee to modify and redistribute the software code as long as the licensor is attributed as the original author and references are made to the original code. *See id.* This practice has also become associated with the term "copyleft," a play on the word copyright. *What is Copyleft?*, GNU Operating Sys., http://www.gnu.org/copyleft/ (last visited Oct. 29, 2010).

a basis to model public licenses to distribute royalty-free software. The dual protection of copyright and patents is therefore unnecessary.[19] Part VI of this note concludes.

II. BACKGROUND

A. Copyright Law

The Copyright Act of 1976 was enacted largely in response to the new and emerging technology at that time, namely "new techniques for capturing and communicating printed matter, visual images, and recorded sounds" and "information storage and retrieval devices, communications satellites, and laser technology."[20] To deal with the "problems raised by the use of the new technologies of photocopying and computers on the authorship, distribution, and use of copyrighted works," Congress created the National Commission on New Technological Uses of Copyrighted Works (CONTU) in an "effort to revise comprehensively the Copyright Laws of the United States."[21] The result was an influential report discussing whether computer software could constitutionally be protected by copyright as "writings" under the Progress Clause[22] and suggesting that Congress amend the Copyright Act to include computer software.[23]

---

19. This dual form of protection also allows a software creator to potentially monopolize an entire field which its software covers. Software is programming code that can be executed to bring about some concept or result, like word processing, for example. While copyright would protect the software developer's "expression" (i.e., the written code) of that result, it would not preclude anyone else from using a substantially different expression of achieving that same result. *See infra* note 181. A software patent, on the other hand, allows the developer to protect any claims to the result, and "[p]atents, by definition, grant the power to exclude others from practicing that which the patent claims." *In re Bilski*, 545 F.3d at 953. Thus, patents would not protect the expression of the software, but the underlying processes or means of how the software achieves its results. These processes or means are essentially an abstract idea or algorithm. This dual protection poses substantial monopoly concerns, but as one can see, a software patent poses other concerns on its own in terms of preempting the use of fundamental concepts that should belong in the public domain. *See infra* notes 146–48 and accompanying text.

20. H.R. Rep. No. 94-1476, at 47 (1976), *reprinted in* 1976 U.S.C.C.A.N. 5659, 5660.

21. Nat'l Comm'n on New Technological Uses of Copyrighted Works, Final Report of the National Commission on New Technological Uses of Copyrighted Works 1 (1978) [hereinafter CONTU Final Report].

22. *See id.* at 36. CONTU determined that:

> [A] program is created, as are most copyrighted works, by placing symbols in a medium. In this respect it is the same as a novel, poem, play, musical score, blueprint, advertisement or telephone directory. It is not the same as a phonorecord or videotape. Those works are created by shaping the physical grooves or electromagnetic fields so that when they are moved past sensing devices electric currents are created which, when amplified, do physical work. Notwithstanding these apparent differences, all these works are writings in the constitutional sense, and eligible for copyright if the Congress so provides.

Id.

23. *See id.*

To support its view that software could and should be eligible for copyright protection, CONTU compared copyright's compatibility with software with other intellectual property protections. CONTU reported that:

> The purpose of copyright is to grant authors a limited property right in the form of expression of their ideas. The other methods used to protect property interests in computer programs have different conceptual bases and, not surprisingly, work in different ways. An appreciation of those differences has contributed to this Commission's recommendation that copyright protection not be withdrawn from programs . . . . Each of these forms of protection may inhibit the dissemination of information and restrict competition to a greater extent than copyright.[24]

Congress ultimately adopted CONTU's recommendations and integrated its statutory proposals into the 1976 Copyright Act.[25] Although software was now statutorily defined in the Copyright Act, there were still problems with certain types of software code that were fundamentally tied to the hardware in a computer like Read Only Memory (ROM). One of the first issues to arise was "whether both source and object code should be protected by copyright, and whether operating system software should be treated the same as application programs" because of how these different codes interacted with the programmer and the computer.[26] It was soon established that source and object code, as well as the code for operating systems, were all literary works under the Copyright Act and subject to copyright protection as original expressions.[27]

24. *Id.* at 40–41.

25. *See supra* note 10 and accompanying text.

26. *See* Cohen et al., *supra* note 6, at 238. *See also* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1248–49 (3d Cir. 1983). Explaining the difference between source and object code and why litigants argued that object code was not subject to copyright, the *Apple Computer* court stated:

> As source code instructions must be translated into object code before the computer can act upon them, only instructions expressed in object code can be used "directly" by the computer. This definition was adopted following the CONTU Report in which the majority clearly took the position that object codes are proper subjects of copyright. The majority's conclusion was reached although confronted by a dissent based upon the theory that the "machine-control phase" of a program is not directed at a human audience.
>
> The defendant in *Williams* had also argued that a copyrightable work "must be intelligible to human beings and must be intended as a medium of communication to human beings."

Id. (quoting Williams Elecs., Inc. v. Arctic Int'l, Inc., 685 F.2d 870, 866–77 (3d Cir. 1982)). Thus, litigants argued that object code, which consists of bits (i.e., binary digits, or zeros and ones), could only be read by computer hardware such as ROM, and therefore, because only a computer could perceive the code and humans could not, copyright protection was unavailable under § 102(a) of the Copyright Act. *Id.* at 1243; *see also* 17 U.S.C. § 102. Source code, on the other hand, consists of the words written by the programmer in a programming language such as Basic or Java, and accordingly, could be perceived by humans. *Apple Computer*, 714 F.2d at 1247.

27. *See supra* note 10. *See also Apple Computer*, 714 F.2d at 1249 ("[T]he category of 'literary works,' one of the seven copyrightable categories, is not confined to literature in the nature of Hemingway's *For Whom the Bell Tolls*. The definition of 'literary works' in section 101 includes expression not only in words but also

In *Computer Associates International, Inc. v. Altai, Inc.*, the Second Circuit announced the "abstraction-filtration-comparison" test "in order to determine whether the non-literal elements of two or more computer programs are substantially similar" for infringement purposes.[28] This test became extremely important because it allowed the courts to determine the protected and unprotected elements of software code. The Second Circuit concluded that "those elements of a computer program that are necessarily incidental to its function" were not protectable under copyright law.[29] The court recognized that "copyright protects computer programs only 'to the extent that they incorporate authorship in [a] programmer's expression of original ideas, as distinguished from the ideas themselves.'"[30] In sum, the Second Circuit offered the following rationale for its proposed test, which demonstrates copyright's compatibility with software:

> In adopting the above three step analysis for substantial similarity between the non-literal elements of computer programs, we seek to insure two things: (1) that programmers may receive appropriate copyright protection for innovative utilitarian works containing expression; and (2) that non-protectable technical expression remains in the public domain for others to use freely as building blocks in their own work. At first blush, it may seem counter-intuitive that someone who has benefited to some degree from illicitly obtained material can emerge from an infringement suit relatively unscathed. However, so long as the appropriated material consists of non-protectable expression, "this result is neither unfair nor unfortunate. It is the means by which copyright advances the progress of science and art[s]."[31]

'numbers, or other . . . numerical symbols or indicia,' thereby expanding the common usage of 'literary works.' Thus a computer program, whether in object code or source code, is a 'literary work' and is protected from unauthorized copying, whether from its object or source code version." (citation omitted)).

28. 982 F.2d 693, 706–12 (2d Cir. 1992). The "non-literal elements" are "those aspects that are not reduced to written code"—for example, how the code is structured. *Id.* at 696. Under this test,

> a court would first break down the allegedly infringed program into its constituent structural parts. Then, by examining each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain, a court would then be able to sift out all non-protectable material. Left with a kernel, or possible kernels, of creative expression after following this process of elimination, the court's last step would be to compare this material with the structure of an allegedly infringing program. The result of this comparison will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of infringement.

Id. at 706.

29. *Id.* at 705.

30. *Id.* at 703 (citation omitted).

31. *Id.* at 721 (citing Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 350 (1991)). *See also* Softel, Inc. v. Dragon Med. & Scientific Commc'ns, Inc., 118 F.3d 955, 964 (2d Cir. 1997) (reinforcing that "'[a]lthough the . . . scrutiny involved in the level-by-level analysis may deny protection to some individual program elements, it must be remembered that a combination of these elements may be protectable. An original arrangement of uncopyrightable or public domain works—even facts—is as copyrightable as a compilation in the computer context as it is elsewhere in copyright law. Thus, individual program elements that are 'filtered' out at one level may be copyrightable when viewed as

B. *Patent Law*

Under the Patent Act of 1952, patents are only granted to inventions and processes that fall under statutory subject matter—in other words, claims can only be patented in categories expressly listed under the Patent Act.[32] As mentioned previously, software is not defined under the Patent Act, but the courts have essentially read it into § 101 as a "process."[33] Paving the way for software as statutory subject matter was the seminal patent case *Diamond v. Chakrabarty*.[34] Although *Chakrabarty* specifically resolved the question of whether a "human-made, genetically engineered bacterium" could be patented as statutory subject matter,[35] the Supreme Court broadly interpreted § 101's subject matter requirement to encompass "anything under the sun that is made by man."[36]

A year later, the Supreme Court decided *Diamond v. Diehr*.[37] The issue in *Diehr* was whether an industrial process using a computer program based on a mathematical algorithm to mold rubber was statutory subject matter.[38] The Court concluded that such a process qualified as subject matter under § 101 and further held that its "conclusion regarding respondents' claims [was] not altered by the fact that in several steps of the process a mathematical equation and a programmed digital computer [were] used."[39] This broad holding was surprising, however, because a decade prior to

part of an aggregate of elements at another level of abstraction.'" (citing Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 Harv. L. Rev. 977, 1003 (1993))).

32. *See supra* note 12. Under § 100 of the Patent Act, an "'invention' means invention or discovery," and a "'process' means process, art, or method, and includes a new use of a known process, machine, manufacture, composition of matter, or material." 35 U.S.C. § 100 (2006). The four categories of statutory subject matter that an invention or process can fall under are process, machine, manufacture, and composition of matter. *Id.*

33. *See supra* note 11.

34. 447 U.S. 303 (1980).

35. *Id.* at 305.

36. *See id.* at 309 (quoting S. Rep. No. 1979-82, at 5 (1952); H. R. Rep. No. 9123-82, at 6 (1952)) (internal quotation marks omitted).

37. 450 U.S. 175 (1981).

38. *See id.* at 177. A main tenet in patent law is that "laws of nature, natural phenomena, and abstract ideas" are unpatentable. *See id.* at 185 (citations omitted). Thus, a common example in patent circles is that Albert Einstein would not be able to receive a patent for "inventing" his mass-energy equivalence formula, $E=mc^2$, since that formula represents a law of nature. *Chakrabarty*, 447 U.S. at 309. Allowing a patent on such a fundamental principle would deprive all inventors from inventing anything based on that formula which would prevent innovation and violate the Progress clause. *See Diehr*, 450 U.S. at 185. Similarly, a mathematical algorithm is also ineligible for patent protection. *See id.* at 186 (defining an "'algorithm' as a 'procedure for solving a given type of mathematical problem,' and . . . such an algorithm, or mathematical formula, is like a law of nature, which cannot be the subject of a patent") (citations omitted). However, "an application of a law of nature or mathematical formula to a known structure or process may well be deserving of patent protection." *Id.* at 187 (citations omitted).

39. *See Diehr*, 450 U.S. at 184–85.

Diehr, the Court in *Gottschalk v. Benson* held that Congress needed to intervene and determine whether software could be patented.[40]

In *Benson*, the Supreme Court rejected, on grounds of non-statutory subject matter, an invention related "'to the processing of data by program and more particularly to the programmed conversion of numerical information' in general-purpose digital computers."[41] Relying on the Report of the President's Commission on the Patent System created in 1966, the Court supported its position by explaining that:

> It is conceded that one may not patent an idea. But in practical effect that would be the result if the formula for converting BCD [binary-coded decimal] numerals to pure binary numerals were patented in this case. The mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself. It may be that the patent laws should be extended to cover these programs, a policy matter to which we are not competent to speak. The President's Commission on the Patent System rejected the proposal that these programs be patentable: "Uncertainty now exists as to whether the statute permits a valid patent to be granted on programs. Direct attempts to patent programs have been rejected on the ground of nonstatutory subject matter. Indirect attempts to obtain patents and avoid the rejection, by drafting claims as a process, or a machine or components thereof programmed in a given manner, rather than as a program itself, have confused the issue further and should not be permitted."[42]

The *Diehr* court read *Benson* as merely reaffirming the established principle that ideas cannot be patented, rather than as a case deferring solely to Congress as to the question of whether a computer program is patentable subject matter.[43] Furthermore, the *Diehr* court distinguished *Benson* on the ground that the process in *Benson* was just a mathematical algorithm, and therefore unpatentable, while the *Diehr* applicants "[did] not seek to patent a mathematical formula. Instead, they [sought] patent protection for a process of curing synthetic rubber."[44] In other words, they were seeking a patent on the *application* of an algorithm.

40. *See* Gottschalk v. Benson, 409 U.S. 63, 73 (1972).

41. *See id.* at 64–73.

42. *See id.* at 71–72 (citation omitted). *See also In re* Johnston, 502 F.2d 765, 774 (C.C.P.A. 1974) (Rich, J., dissenting) (stating that *Benson* would direct the court to affirm the rejection of claims for computer software as directed towards non-statutory subject matter under § 101 of the Patent Act), *rev'd by* Dann v. Johnston, 425 U.S. 219 (1976). In *Dann*, the Supreme Court reversed the majority's decision in *Johnston* which held that the claims for computer software, specifically a computer program that performed record-keeping functions for banks, could receive a patent. 425 U.S. at 220. The Court, however, refused to decide the issue of whether the computer software was statutory subject matter, and instead, held that the claims for the software were invalid because they were obvious, one of the threshold questions for patentability under § 103 of the Patent Act. *Id.*

43. *See Diehr*, 450 U.S. at 185; *see also supra* note 38.

44. *See Diehr* 450 U.S. at 187.

Of course, Congress never engaged in a broad investigation into whether software could or should qualify as statutory subject matter under the Patent Act after the Supreme Court's recommendation in *Benson*. *Diehr* and its progeny, however, established precedent for the claim that software was indeed subject matter worthy of patent protection under § 101.

In 1998, the Federal Circuit was faced with a question concerning the validity of a patent "generally directed to a data processing system . . . for implementing an investment structure."[45] Upholding the validity of the patent and determining that the associated claims[46] were directed to statutory subject matter under § 101,[47] the *State Street* court provided further support for the subject matter eligibility of software.

---

45. State Street Bank & Trust Co. v. Signature Fin. Group, 149 F.3d 1368, 1370 (Fed. Cir. 1998), *abrogated by In re* Bilski, 545 F.3d 943 (Fed. Cir. 2008).

46. Claim 1 of the *State Street* patent recited the following:

> 1. A data processing system for managing a financial services configuration of a portfolio established as a partnership, each partner being one of a plurality of funds, comprising:
>
> (a) computer processor means [a personal computer including a CPU] for processing data;
>
> (b) storage means [a data disk] for storing data on a storage medium;
>
> (c) first means [an arithmetic logic circuit configured to prepare the data disk to magnetically store selected data] for initializing the storage medium;
>
> (d) second means [an arithmetic logic circuit configured to retrieve information from a specific file, calculate incremental increases or decreases based on specific input, allocate the results on a percentage basis, and store the output in a separate file] for processing data regarding assets in the portfolio and each of the funds from a previous day and data regarding increases or decreases in each of the funds, [sic] assets and for allocating the percentage share that each fund holds in the portfolio;
>
> (e) third means [an arithmetic logic circuit configured to retrieve information from a specific file, calculate incremental increases and decreases based on specific input, allocate the results on a percentage basis and store the output in a separate file] for processing data regarding daily incremental income, expenses, and net realized gain or loss for the portfolio and for allocating such data among each fund;
>
> (f) fourth means [an arithmetic logic circuit configured to retrieve information from a specific file, calculate incremental increases and decreases based on specific input, allocate the results on a percentage basis and store the output in a separate file] for processing data regarding daily net unrealized gain or loss for the portfolio and for allocating such data among each fund; and
>
> (g) fifth means [an arithmetic logic circuit configured to retrieve information from specific files, calculate that information on an aggregate basis and store the output in a separate file] for processing data regarding aggregate year-end income, expenses, and capital gain or loss for the portfolio and each of the funds.

Id. at 1371–72. This type of claim is known as a "means-plus-function" claim. *See* 35 U.S.C. § 112 (2006) (allowing claims to include elements "as a means or step for performing a specified function"). *See generally* Kieff et al., *supra* note 9, at 94–96 ("Such a claim element defines the function of the element, rather than the structure.").

47. *State Street*, 149 F.3d at 1370.

At trial, the *State Street* district court had held that the patent was invalid under "judicially-created exceptions" to § 101, namely the "mathematical algorithm" and "business method" exceptions.[48] The Federal Circuit in *State Street* expressly rejected these bright-line rules and held

> 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.[49]

As a result, this ruling meant that computer programs, which are fundamentally implementations of algorithms,[50] are statutory subject matter as long as the program results in something "useful, concrete and tangible."[51]

In 2008, the Federal Circuit again altered its statutory subject matter test—specifically as it relates to business method patents that rely on computer software to implement underlying processes—in a landmark case involving "a method of hedging risk in the field of commodities trading."[52] The Federal Circuit in *Bilski* framed the issue as follows: "The question before us then is whether Applicants' claim recites a fundamental principle, and, if so, whether it would pre-empt substantially all uses of that fundamental principle if allowed."[53] In answering this question, the court abrogated the *State Street* "useful, concrete and tangible" test[54] and returned to principles that the court derived from *Benson* and *Diehr*. From these two precedents, the court enunciated the "machine-or-transformation" test, which states that a process or method that uses an underlying mathematical algorithm can qualify as statutory subject matter only if it is tied to a machine or if it transforms something into a "different state or thing."[55] For example, the *Diehr* "process operated on a computerized rubber curing apparatus and transformed raw, uncured rubber into molded, cured rubber products," and was therefore statutory subject matter.[56]

48. *Id.* at 1372.

49. *Id.* at 1373 (quoting *In re* Alappat, 33 F.3d 1526, 1544 (Fed. Cir. 1994)).

50. *See* Diamond v. Diehr, 450 U.S. 175 (1981); *see also* Samuelson et al., *supra* note 9, at 2321 n.37 ("An algorithm is 'a prescribed set of well-defined, unambiguous rules or processes for the solution of a problem in a finite number of steps . . . .").

51. *See State Street*, 149 F.3d at 1374 (quoting *In re Alappat*, 33 F.3d at 1544). In *State Street*, the result which would be generated by the patent was just a number. *See id.* at 1375 ("This renders [Claim 1] statutory subject matter, even if the useful result is expressed in numbers, such as price, profit, percentage, cost or loss.").

52. *In re Bilski*, 545 F.3d at 949.

53. *Id.* at 954.

54. *Id.* at 959–60 n.19.

55. *Id.* at 954.

56. *Id.*

Ultimately, the court found that the method for hedging risk was directed to non-statutory subject matter and could not be patented.[57]

The Federal Circuit's machine-or-transformation test in *Bilski* has resulted in numerous rejections of subsequent patent applications for software processes by the Board of Patent Appeals and Interferences (BPAI).[58] The Federal Circuit, however, specifically "decline[d] to adopt a broad [statutory subject matter] exclusion over software."[59] The Supreme Court heard the *Bilski* appeal in its October Term, 2009—an appeal that was closely watched by those in the software industry.[60] Although the Court affirmed the Federal Circuit's judgment rejecting the petitioners' business method patent application as a non-patentable process,[61] the Court did not address the issue of the validity of software patents under § 101 of the Patent Act. Rather, it only held that the machine-or-transformation test was not the exclusive test for determining "whether an invention is a patent-eligible 'process.'"[62] Thus, the question of whether claims based on software are directed towards statutory subject matter remains an open one, and one that the Court will eventually have to decide. Meanwhile, the implications of *Bilski* for the future remain to be seen.

C. *The Theories Behind Software Patents*

Patents, like copyrights, are justified on utilitarian grounds.[63] Based on these principles, if inventors do not receive patent protection for their inventions, they will

57. *Id.* at 965–66. Applying the machine-or-transformation test, the court concluded that:

> Applicants here seek to claim a non-transformative process that encompasses a purely mental process of performing requisite mathematical calculations without the aid of a computer or any other device, mentally identifying those transactions that the calculations have revealed would hedge each other's risks, and performing the post-solution step of consummating those transactions. Therefore, claim 1 would effectively pre-empt any application of the fundamental concept of hedging and mathematical calculations inherent in hedging (not even limited to any particular mathematical formula). And while Applicants argue that the scope of this pre-emption is limited to hedging as applied in the area of consumable commodities, the Supreme Court's reasoning has made clear that effective pre-emption of all applications of hedging even just within the area of consumable commodities is impermissible. Moreover, while the claimed process contains physical steps (initiating, identifying), it does not involve transforming an article into a different state or thing.

Id.

58. *See supra* note 14 and accompanying text.

59. *In re Bilski*, 545 F.3d at 960 n.23.

60. *See supra* note 15.

61. Bilski v. Kappos, 130 S. Ct. 3218, 3231 (2010).

62. *Id.* at 3227.

63. *See generally* Edwin C. Hettinger, *Justifying Intellectual Property*, 18 Phil. & Pub. Aff. 31, 48 (1988) ("If competitors could simply copy books, movies, and records, and take one another's inventions and business techniques, there would be no incentive to spend the vast amounts of time, energy, and money necessary to develop these products and techniques . . . . Granting property rights to producers is here seen as necessary to ensure that enough intellectual products . . . are available to users."); Paula Baron,

be less likely to contribute to progressive endeavors that ultimately add to the inventory of public knowledge. Accordingly, patents serve as economic incentives for inventors to create new and useful inventions. Once a patent is granted for an invention, the inventor can exclude people from using or selling it,[64] giving him bargaining power to license his invention to others for monetary consideration. In return, however, the inventor must publicly disclose his invention in his patent application[65] and is only granted a limited term for his monopoly over it.[66] Once the term of the patent expires, the invention becomes part of the public domain.

Utilitarian principles are primarily used to justify protecting software under the Patent Act.[67] Without patent rights, software developers do not have economic incentives to create the important programs that run our computers and help our lives. If people could just freeload off their creations, they would be unable to recoup their expenses or earn a return from their development investments. Ultimately, software development would come to a halt and there would be fewer software products on the market. Software patents, however, purportedly solve this public goods problem, prevent market failure, promote progress, and spur innovation.[68]

The Moebius Strip: Private Right and Public Use in Copyright Law, 70 Alb. L. Rev. 1227, 1238 (2007) (defining the "underlying utilitarian rationale for copyright protection [as] encouraging the production of new and useful works").

64. 35 U.S.C. § 154(a)(1) (2006) (stating that inventors can also exclude someone from making or offering to sell their patented invention).

65. 35 U.S.C. § 112 (2006) (stating that the invention must be disclosed to the extent that it "enables[s] any person skilled in the art" of the patent to make it).

66. *See generally* 35 U.S.C. § 154 (2006).

67. *See, e.g.*, Bradford L. Smith & Susan O. Mann, *Innovation and Intellectual Property Protection in the Software Industry: An Emerging Role for Patents?*, 71 U. Chi. L. Rev. 241, 241 (2004) (describing how intellectual property protection provides incentives for software developers to invest in developing new programs and thus promotes progress of the field).

68. *See generally* Mark A. Lemley, *Antitrust and the Internet Standardization Problem*, 28 Conn. L. Rev. 1041, 1053 (1996). As Professor Lemley explained:

> [T]he ease of imitation of software in the absence of a legal regime preventing such copying suggests that a market for operating systems where copying was permitted would be competitive—firms would sell programs at their marginal cost of copying, probably for less than $1 each. However, this low marginal cost would prevent the first developer of an operating system from recouping its initial fixed costs of designing and producing the program, and would therefore discourage subsequent developers from producing new systems. It is this "public goods" problem which justifies intellectual property protection for software.

Id. See also Wendy J. Gordon, *Assertive Modesty: An Economics of Intangibles*, 94 Colum. L. Rev. 2579, 2587–88 (1994) ("[T]here are at least two levels at which markets can fail to foster appropriate cost/benefit tradeoffs. At the first level, public goods—things that can be shared by many without physical diminution, and for which it is difficult to exclude nonpayors—can give rise to a pattern in which consumers will get less of the good than they would otherwise be willing to pay for. Such market failures can be costly enough to justify the law in imposing restraints on copying, such as the law of copyright, patent, and unfair competition."); Carol Rose, *The Comedy of the Commons: Custom, Commerce, and Inherently Public Property*, 53 U. Chi. L. Rev. 711, 718–19 (1986) ("Since the mid-nineteenth century, economists have told us that there exist predictable instances of 'market failure,' where Adam

Proponents of software patents theorize that these patents help to facilitate progress in other ways as well. They argue that the public disclosure requirement of the Patent Act[69] gives other software developers information to develop new software inventions (based on the underlying ideas in the previous invention) without infringing on any of the disclosed claims of which the new inventor is aware and can now avoid duplicating. As Microsoft Corporation executives Smith and Mann argue:

> Patents seek to promote technological progress by giving exclusive rights in discrete inventions in exchange for early public disclosure of the invention. Exclusivity gives the innovator control over the patented invention. This, in turn, enables the patent owner to realize economic benefits, either through sales or licensing. Exclusivity provides both an economic incentive for the initial invention and its commercial development, as well as a stimulus for the development of new, noninfringing technology through other independent inventions or design-arounds.[70]

They also argue that the patent term[71] is far shorter than the respective terms in copyright[72] or trade secret (which is potentially unlimited),[73] and, therefore, the protected invention is outside of the public domain for less time.[74] Despite the merit of these theories, software patents ultimately do not promote progress within the software industry. As discussed in the following part of this note, strong empirical and anecdotal evidence demonstrates that the negative effects of software patents simply cannot be reconciled with these traditional patent theories.

III. PATENTS RESTRICT SOFTWARE DEVELOPMENT

Since the Supreme Court first upheld the validity of software patents, substantial statistical research and anecdotal evidence has suggested that these patents may negatively influence the software industry.[75] A software patent is a legal monopoly that gives a software inventor a limited time to exclude others from making, using,

Smith's invisible hand fails to guide privately owned resources to their socially optimal uses. These involve 'public goods,' 'natural monopolies,' 'externalities,' and the like.").

69. *See supra* note 65 and accompanying text.

70. *See* Smith & Mann, *supra* note 67, at 256–57.

71. Generally, the term of a utility patent issuing from an application filed under 35 U.S.C. § 111(a) is twenty years from the filing date of the application. *See* 35 U.S.C. § 154(a)(2) (2006).

72. Computer software is often created as a work made for hire. As a work made for hire created on or after January 1, 1978, "the copyright endures for a term of 95 years from the year of its first publication, or a term of 120 years from the year of its creation, whichever expires first." 17 U.S.C. § 302(c) (2006). Generally, if the software is created on or after January 1, 1978 and is not a joint work, an anonymous work, a pseudonymous work, or a work made for hire, but perhaps independently developed, then the copyright would last for "the life of the author and 70 years after the author's death." *Id.* § 302(a).

73. *See* Craig Allen Nard, David W. Barnes & Michael J. Madison, The Law of Intellectual Property, 937 (2d ed. 2008).

74. *See* Smith & Mann, *supra* note 67, at 257.

75. *See supra* note 9 and accompanying text.

or selling their claimed software without some prior, agreed-upon consideration.[76] As discussed in more detail below, this exclusive right can limit a software developer's ability to innovate, and halt progress.

A. Decreased Investments in Research and Development

One area in which patents have negatively impacted the software industry is in research and development (R&D). Empirical data suggests that during the 1990s, after software patents were becoming widely accepted, R&D investments actually declined in firms that engaged in software development when instead they should have been rising as a result of the increased patenting of software.[77] In their statistical analysis, Bessen and Hunt assumed that the "incentive hypothesis," or utilitarian justification, means "that R&D and patents are complements" and that "increases in the appropriability of software should lead to greater R&D intensity."[78] Instead, they found that the opposite occurred and that, from 1991 to 1997, software organizations were increasing their patent portfolios while reducing investments in R&D.[79] Although causal relationships could not be identified, this study strongly implies that software patent owners may have found it more cost-effective to generate revenue

76. *See supra* note 64 and accompanying text. *See also supra* note 19 (explaining the difference between a copyright monopoly and a patent monopoly).

77. *See* James Bessen & Robert M. Hunt, *An Empirical Look at Software Patents* 4 (Fed. Bank of Phila., Working Papers 03-17, 2004). As discussed in Part II.C, the utilitarian theory behind patent protection for software predicts increased R&D activity because software developers will want to invest in endeavors that will produce new and innovative software as a result of the economic benefits they will receive in return from the incentives created by more cost-effective patent rights. *See id.* at 26.

78. Bessen & Hunt, *supra* note 77, at 4. A primer of Bessen's and Hunt's method for their analysis is in order. Appropriability "refers to the environmental factors, excluding firm and market structure, that govern an innovator's ability to capture the profits generated by an innovation." David J. Teece, *Profiting from Technological Innovation: Implications for Integration Collaboration, Licensing and Public Policy*, *in* ESSAYS IN TECHNOLOGY MANAGEMENT AND POLICY SELECTED PAPERS OF DAVID J. TEECE 15, 28 (2003). Thus, patents are a means of appropriation for companies and inventors. Instead of relying on the U.S. Patent and Trademark Office's classifications for software patents for their data, Bessen and Hunt "perform[ed] a keyword search [based on an algorithm they developed] of the U.S. Patent Office database, which identified 130,650 software patents granted in the years 1976 to 1999." Bessen & Hunt, *supra* note 77, at 8. Their definition of a "software patent involve[d] a logic algorithm for processing data that is implemented via stored instructions; that is, the logic is not 'hard-wired.'" *Id.* They then derived a regression equation to study the correlation between two variables: R&D costs and percentage of software patents in a company's patent portfolio. *Id.* at 28–29. "Regression analysis is a statistical tool for the investigation of relationships between variables." Alan O. Sykes, *An Introduction to Regression Analysis* 1 (Univ. of Chi. L. Sch. John M. Olin Program in Law & Econ., 2d Ser., Working Paper No. 20, 1993), *available at* http://www.law.uchicago.edu/files/files/20.Sykes_.Regression.pdf; BESSEN & MEURER, *supra* note 5, at 82 ("Multiple regression analysis is a statistical technique used when researchers want to analyze a phenomenon that might be associated with multiple independent factors.").

79. *See* Bessen & Hunt, *supra* note 77, at 30–33. *See also* Ben Klemens, *The Current State of Software and Business Method Patents: 2008 Edition*, END SOFTWARE PATENTS 1, 5 (2008), *available at* http://esp.wdfiles.com/local--files/2008-state-of-softpatents/feb_08-summary_report.pdf (finding that academic researchers who studied software patents granted in the 1990s were unable to find any empirical evidence that supported a correlation between increased innovation and software patents).

from existing inventions by building up and exploiting their patent portfolios, rather than by engaging in more R&D to create new software.[80]

In fact, anecdotal evidence supports this trend of investing in pre-existing inventions rather than R&D and explains some of the reasons behind it. In an October 2003 report, the Federal Trade Commission (FTC) found that:

> Much of this thicket of overlapping patent rights results from the nature of the technology; computer hardware and software contain an incredibly large number of incremental innovations. Moreover, as more and more patents issue on incremental inventions, firms seek more and more patents to have enough bargaining chips to obtain access to others' overlapping patents. One panelist asserted that the time and money his software company spends on creating and filing these so-called defensive patents, which "have no . . . innovative value in and of themselves," could have been better spent on developing new technologies.[81]

These defensive (or strategic) patents are common in the software industry. One hypothesis for this defensive use of software patents is that "[m]aturing firms with diminished competitive advantage from technology might choose to harvest patent royalties from their past research in lieu of further R&D, especially if legal changes make patents more cost effective."[82] Indeed, the U.S. Patent and Trademark Office (USPTO) has issued approximately 200,000 software patents to date,[83] perhaps evidencing an increase in the value of obtaining patents for software developers. Another more probable explanation is based on the nature of the software industry. Software development is generally considered "sequential," meaning that new software is produced using previous (usually protected) software and ideas.[84] This is also characterized as "incremental innovation."[85]

The evidence gathered suggests that in the software industry this sequential process involving the associated burden of "patent thickets"[86] is slowed down by software patents, which in turn, discourage software innovation. In the FTC Report,

80. *See* Bessen & Hunt, *supra* note 77, at 38–40. This process has been referred to as "strategic patenting." *See id.* at 40 (citation omitted).

81. Fed. Trade Comm'n, To Promote Innovation: The Proper Balance of Competition and Patent Law and Policy, Exec. Summary at 6–7 (2003) [hereinafter FTC Report].

82. Bessen & Hunt, *supra* note 77, at 39.

83. *See* Bessen & Meurer, *supra* note 5, at 22.

84. *See id.* at 4.

85. *See* FTC Report, *supra* note 81, Exec. Summary at 6 ("In some industries, such as computer hardware and software, firms can require access to dozens, hundreds, or even thousands of patents to produce just one commercial product . . . . Many of these patents overlap, with each patent blocking several others. This tends to create a 'patent thicket'—that is, a 'dense web of overlapping intellectual property rights that a company must hack its way through in order to actually commercialize new technology.'"). *See also* Vivek Wadhwa, *Why We Need to Abolish Software Patents*, TechCrunch, (Aug. 7, 2010), http://techcrunch.com/2010/08/07/why-we-need-to-abolish-software-patents (anecdotally noting that competitors can learn from a software patent filing to "do things better").

86. FTC Report, *supra* note 81, at 6.

"panelists from the software industry complained of the risk of hold-up, noting that the owner of any one of the multitude of patented technologies constituting a software program can hold up production of innovative new software."[87] Thus, while building upon previous inventions is a natural process in other industries, in the software industry, rights owners tend to use their patents mainly "as bargaining chips in cross-licensing negotiations."[88] As a result, other inventors are prevented from developing new software technology based on previous ones if negotiations fail, or they are forced to redirect their R&D funds toward engaging in this expensive process.[89] These restrictive uses of software patents not only interrupt innovation, but also create significant infringement risks and costs for subsequent software developers such that "'a second innovator may choose to perform a sub-optimal level of R&D or, perhaps, not to invest in the innovation at all.'"[90]

B. Increased Litigation & Risks

There is also substantial evidence showing that the existence of software patents increases litigation and the risk of infringement claims. In fact, software patents have a 4.6% likelihood of being involved in a lawsuit, the second highest among all categories of technological patents.[91] These figures are particularly problematic for

87. *Id.* ch. 2, at 3. Furthermore, one panelist who participated in the FTC Report "issued a directive to his company requiring that they 'reallocate roughly 20 to 35 percent of [their] develop[ment] resources . . . [in order to] sign on two separate law firms to increase [their] patent portfolios for purely defensive reasons." *Id.* ch. 3, at 52. Obviously, this resulted in decreased R&D resources. This statement was made by R. Jordan Greenhall, co-founder and former CEO of DivX, Inc., the developers of a computer program that allows internet users to view high-resolution video streams. *See id.* n.343; *id.* app. A, at A-8.

88. *See* FTC Report, *supra* note 81, ch. 3, at 33–34; *see also* Wadhwa, *supra* note 85 ("[I]n software[, patents] are just nuclear weapons in an arms race. They don't foster innovation, they inhibit it. That's because things change rapidly in this industry. Speed and technological obsolescence are the only protections that matter. Fledgling startups have to worry more about some big player or patent troll pulling out a big gun and bankrupting them with a frivolous lawsuit than they do about someone stealing their ideas.").

89. *See generally* Bessen & Maskin, *supra* note 9, at 1–6 (finding that in industries "in which innovation is both *sequential* and *complementary*," like the software industry, "strong patents become an impediment" to innovation; sequential "mean[s] that each successive invention builds on the preceding one, in the way that the Lotus 1-2-3 spreadsheet built on VisiCalc, and Microsoft's Excel built on Lotus," while complementary "mean[s] that each potential innovator takes a different research line and thereby enhances the overall probability that a particular goal is reached within a given time"); *see also* FTC Report, *supra* note 81, ch. 3, at 1–2 (noting that over the course of a six-day hearing, business representatives from the computer hardware and software industry generally "discussed how patent thickets drive funds away from R&D, make it difficult to commercialize new products, and raise uncertainty and investment risks").

90. *See* FTC Report, *supra* note 81, ch. 3, at 50–51 (citations omitted).

91. *See* Bessen & Meurer, *supra* note 5, at 153. Business method patents have the highest likelihood at 13.7%. *Id.*

the software industry because the annual litigation cost for software patents is much higher than the profits that these patents generate.[92]

Furthermore, there is no sign that litigation associated with software patents is declining or even slowing down. Indeed, the opposite seems to be occurring. From 1984 to 2002, Professors Bessen and Meurer found that the "percentage of patent lawsuits involving software patents" had increased from less than 5% in 1984 to a staggering 26% in 2002.[93] Their empirical evidence further demonstrates that a software patent is more likely to be the subject of litigation within four years of its issue than all other patents, and this probability has been steadily increasing since 1984.[94]

Anecdotal evidence provides additional support for this empirical data that shows software patents causing increased litigation and risks in what is truly a circular cycle. Along with decreased R&D investments, defensive software patenting is one of the causes behind this phenomenon: "As more patents issue, the likelihood of 'unintentional and sometimes unavoidable patent infringement' increases. Some firms respond to this by 'fil[ing] hundreds of patents each year' themselves, patents they can use defensively against firms threatening infringement actions. The result of this, of course, is yet more patenting."[95] And, as a likely consequence, more lawsuits arise from infringement claims related to software patents, which also "impose higher litigation costs than other types of patents."[96]

The exact reason behind the increased litigation and infringement risks that software patents create is uncertain.[97] The impact this has on the software industry, however, is not. Increased threats of litigation from patent infringement claims, as well as a high probability of actually being involved in a patent lawsuit, contribute to

92. *See id.* at 143–44 (finding that from 1996 to 1999 the aggregate annual U.S. litigation costs for software patents was $3.88 billion in 1992 dollars, while the aggregate annual U.S. profits for software patents was only $100 million in 1992 dollars).

93. *See* Bessen & Meurer, *supra* note 5, at 192 fig. 9.1. These results also suggest "that software technology might be driving some of the growth in litigation" in patents overall because "[t]he two industries with the highest growth rates in litigation are both heavy users of software: namely, business services/software and machinery/computers . . . ." *Id.* at 156–57.

94. *See id.*, at 193. The researchers also found that "[s]oftware patents issued in more recent years are much *more* likely to be litigated, not less." *Id.* (emphasis added).

95. FTC Report, *supra* note 81, ch. 2, at 26–27.

96. Bessen & Meurer, *supra* note 5, at 194.

97. *See*, *e.g.*, *id.* at 155, 157, 187, 194 (arguing that "software is an *abstract* technology," and therefore, "software patents suffer notice problems [from inventors being unable to determine the metes and bounds of the software patent]" and "have unclear boundaries," which results in "opportunistic litigation" as well as higher costs when litigation ensues; and finding that "notice problems explain a wide variety of evidence about litigation rates over time and across technologies. This makes the notice function a strong candidate to explain the large increase in litigation risk that remains after measurable factors have been take into account"); FTC Report, *supra* note 81, ch. 3, at 9, 52–53 (reporting that software patent thickets make avoiding patent infringement difficult; furthermore, stating that panelists participating in the report found that "the PTO issues too many questionable patents [generally arising from the PTO granting patents that are broader than their enablement or that do not necessarily meet the requisite elements of patentability such as non-obviousness], which create a gridlock of patent litigation in the district court system").

decreased R&D spending; higher market-entry costs; uncertainties relating to business decisions; and "scar[ing] away venture capital."[98] Thus, software patents, with their high litigation rates and infringement risks, create a perilous environment for those who try to develop innovative software.

IV. AN INCOMPATIBLE PATENT SYSTEM

This Part argues that the problems discussed in Part III are the result of patent norms that are simply unsuited to deal with software. Because of this incompatibility, many software patents are granted to the detriment of the software industry. These patents result in the diversion of funds away from R&D expenditures[99] and imposition of higher costs and risks on software developers.[100] Software developers are ultimately hampered from making progress, or worse, prevented from creating new software altogether.

In applying for a patent, both the patent examiner and applicant must identify prior art relevant to their invention in order to determine whether the invention meets the patentability requirements of novelty and non-obviousness.[101] In a

98. *See* FTC Report, *supra* note 81, ch. 5, at 2–4. *See also* Wayne M. Kennard, *Software Patents and the Internet*, *in* Practising Law Institute: Fourth Annual Internet Law Institute 311, 313–14 (PLI Patents, Copyrights, Trademarks, and Literary Property Course Handbook Ser. No. G0-00D6, 2000). Kennard explained that:

> Many companies, even those accustomed to using patents to protect their intellectual property, feel patent litigation diverts tremendous amounts of financial and human resources away from their core business, resources that could be better used on research and development, company expansion, or a million other things. This theme is echoed even more loudly by software companies (that are not accustomed to using patents to protect their intellectual property) because, as an industry, software companies have had a strong anti-patent bias. This bias, in large part, is based on many software engineers' belief that the software they have developed or has been developed by others is *not patentable*.

Id.

99. *See supra* Part III.A.

100. *See supra* Part III.B.

101. "Prior art" is a term of art in patent law that goes hand-in-hand with the Patent Act's requirement of novelty (the invention must be new) and nonobviousness (the invention must not have been obvious to a person having ordinary skill in the art of the invention's subject matter) under § 102 and § 103, respectively. *See* Kieff et al., *supra* note 9, at 323–26, 531–36 (discussing novelty and nonobviousness). Prior art can encompass a previous patent application, an old invention (patented or unpatented), or non-patent references such as an article in an academic journal. *See id.* at 324–26. *See also* FTC report, *supra* note 81, at 9 n.27 ("'[P]rior art' consists of materials—often patents and publications, although affidavits and testimony also may present prior art—that reflect one or more of the features or elements of the claimed invention. An invention is 'obvious' if it does not represent a sufficient step beyond the prior art."). "If an invention isn't new, it is said to be *anticipated* by prior art," and is not patentable. Kieff et al., *supra* note 9, at 323. Prior art can also be combined to make a new invention obvious, and therefore, unpatentable. *See generally* 35 U.S.C. § 103 (2006); Kieff et al., *supra* note 9, at 326. "In examining the application, the Examiner will determine whether the invention is novel (according to 35 U.S.C. § 102) and nonobvious (according to 35 U.S.C. § 103) in view of the prior art." Kennard, *supra* note 98, at 318.

normative patent system, a patent examiner would have most, if not all, of the relevant prior art in front of him for consideration.[102] This is far from the case, however, with software patent applications. Industry practices, as well as the abstract nature of software technology,[103] make researching the prior art for software inventions difficult,[104] which results in inadequate prior art findings.[105]

Anecdotal evidence suggests that this difficulty occurs because patent examiners do not have enough time to research all the relevant software prior art, and the USPTO has limited funds to engage in more thorough searches.[106] The patent examiners who are assigned to software inventions may also contribute to these poor prior art searches.[107] However, much of these prior art problems can be attributed to the fact that most software developers who operate outside of large organizations choose not to patent their software.[108] The result is a large gap in the amount of prior

102. *See* John R. Allison & Mark A. Lemley, *The Growing Complexity of the United States Patent System*, 82 B.U. L. Rev. 77, 101 (2002) ("Citations to prior art are an important proxy for the rigor of the examination process.").

103. *See* Diamond v. Diehr, 450 U.S. 175, 177, 184, 187 (1981) (discussing abstract ideas and algorithms); *see generally* Bessen & Meurer, *supra* note 5, at 186 (arguing that "software is an *abstract* technology"). Bessen and Meurer use this term as meaning "abstract ideas or principles." Bessen & Meurer, *supra* note 5, at 187.

104. *See* Kennard, *supra* note 98, at 326 ("In other than software cases, the Applicant's and Examiner's patent searches can be relied on to provide some of the best prior art that would apply to the patentability of an invention. This assumption cannot be made with respect to the software applications."); FTC Report, *supra* note 81, at 45–46 ("The formal recognition of the patentability of software . . . has spurred increased patenting and has presented challenges in locating the relevant prior art, much of which exists outside of traditional prior art sources."); Bessen & Meurer, *supra* note 5, at 198–99 (explaining that abstract software claims make it "difficult . . . to determine the content of prior art").

105. *See* Kennard, *supra* note 98, at 322 (finding that in 1998 the average number of prior art references in a software patent was eleven, which is considered a small number of references).

106. *See* FTC Report, *supra* note 81, at 9–10 n.35 ("The patent prosecution process involves only the applicant and the PTO. A patent examiner conducts searches of the relevant prior art, a focal point of the examination process, with only the applicant's submissions for assistance . . . . Hearings participants estimated that patent examiners have from 8 to 25 hours to read and understand each application, search for prior art, evaluate patentability, communicate with the applicant, work out necessary revisions, and reach and write up conclusions. Many found these time constraints troubling. Hearings participants unanimously held the view that the PTO does not receive sufficient funding for its responsibilities."). *See also* Kennard, *supra* note 98, at 322 ("In fact, as of 1998, statistically, you would find that the average number of references cited is approximately eleven references per patent. Of these eleven, eight are U.S. patents, one is a foreign patent, and two are non-patent references. These numbers indicate that U.S. patent searches by Applicants or Examiners are the primary source of prior art.").

107. *See* Kennard, *supra* note 98, at 325 ("Although the Examiners who handle the prosecution of software patents may be very accomplished and knowledgeable in hardware, these same Examiners usually do not have the same level of knowledge and understanding of software. This issue has much to do with the ability of the Patent Office to properly handle the prosecution of the software-related applications and consider the prior art that is applicable to such applications.").

108. *See id.* at 317 ("[S]oftware companies and vendors as a whole are not particularly interested in seeking patents to protect their inventions . . . ."); Bessen & Meurer, *supra* note 5, at 189–90; *see also* FTC Report, *supra* note 81, ch. 3, at 54 (finding that "this lack of adequate consideration of prior art is attributable" to "(1) the informal nature of software development, especially among the open source

art references available to both the software patent examiner and applicant. Moreover, these non-patent references are difficult to find.[109] In fact, experts suggest that *most* software prior art is part of this inventory of non-patent references.[110] Yet, the majority of a software patent's prior art references typically consist of previous patent references with minimal non-patent references.[111] The result is an issued software patent that may not be novel or that is obvious.[112] These patents, in turn, lead to many of the problems identified in Part III of this note.

community; (2) the rapidly changing and complex nature of the software and Internet industries; (3) the absence of a legal requirement for patent applicants to disclose source code; (4) the use of trade secrecy for almost 20 years of commercial software development; and (5) the relatively recent recognition of the validity of business method patents by the courts").

109. Kennard, *supra* note 98, at 323 ("The search for non-patent prior art is more difficult. In part, this is because there is no central depository that can be searched.").

110. *See* Robert P. Merges, *As Many as Six Impossible Patents Before Breakfast: Property Rights for Business Concepts and Patent System Reform*, 14 Berkeley Tech. L.J. 577, 589 (1999) ("There is every reason to believe that there is a vast volume of non-patent prior art in the software-implemented business concept field, as is widely believed to be the case with software patents in general."); Allison & Lemley, *supra* note 102, at 102 ("This absence of non-patent prior art is particularly striking, given that in many areas of technology, existing or prospective patents may not be the best source of prior art."); Julie E. Cohen, *Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implications of "Lock-Out" Programs*, 68 S. Cal. L. Rev. 1091, 1178 (1995) ("[I]n the field of computers and computer programs, much that qualifies as prior art lies outside the areas in which the PTO traditionally has looked—previously issued patents and previous scholarly publications. Many new developments in computer programming are not documented in scholarly publications at all. Some are simply incorporated into products and placed on the market; others are discussed only in textbooks or user manuals that are not available to examiners on line. In an area that relies so heavily on published, 'official' prior art, a rejection based on 'common industry knowledge' that does not appear in the scholarly literature is unlikely. Particularly where the examiner lacks a computer science background, highly relevant prior art may simply be missed.").

111. *See* Kennard, *supra* note 98, at 322 ("If a random selection of software patents is reviewed, it will be quickly noticed that the average number of prior art references cited in each is small. In fact, as of 1998, statistically, you would find that the average number of references cited is approximately eleven references per patent. Of these eleven, eight are U.S. patents, one is a foreign patent, and two are non-patent references. These numbers indicate that U.S. patent searches by Applicants or Examiners are the primary source of prior art."). Kennard also found that the number of non-patent references in a software patent application has declined since 1998. *See id.* at 325 n.10.

112. Julie E. Cohen & Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 Calif. L. Rev. 1, 12–13 (2001) ("Abundant evidence indicates that the PTO has issued software patents on a number of applications that did not meet the standard tests of novelty and nonobviousness."); Dan L. Burk & Mark A. Lemley, *Is Patent Law Technology-Specific?*, 17 Berkeley Tech. L.J. 1155, 1169–70 (2002) ("[L]egions of scholars and commentators complain that the PTO is issuing too many software patents, and in particular that it is issuing patents on subject matter that should be considered obvious. We agree with these commentators that the PTO is issuing bad software patents, in part because it cannot find relevant prior art. But our point is a different one: those patents will not fare well in litigation because the Federal Circuit will consider them obvious in view of any other computer program that implements the same basic concepts, regardless of how different those programs are in detailed implementation, or perhaps even in view of prior art merely suggesting the desirability of such a program. Further, while hidden prior art is indeed a problem, parties in litigation have far more time and money to spend than do patent examiners, and they are much more likely than the PTO to find the best prior art. The probable result is that, while numerous software patents will issue, a large number of those actually litigated will be found obvious and thus invalid.").

Furthermore, this poses a huge risk for a software developer who is investing substantial sums of money into the patent process.[113] In the event that relevant prior art is not found, a software patent that has been issued may later be deemed invalid during an infringement trial when the prior art is subsequently discovered.[114] The obvious consequence is that the patent owner would lose the investment in the patent and incur considerable litigation costs that could have been avoided had the prior art been found during the application process. More problematic, however, is that inadequate prior art disclosures may also cause a software patent to be issued with a "broader scope."[115] Instead of being invalidated in an infringement proceeding, as it should be, a patent issued without consideration of the *entire* relevant prior art will

113. The cost of trying to obtain a software patent from the USPTO, and then maintaining it, is expensive. *See* Kennard, *supra* note 98, at 326 (finding that "the average costs for *preparing* software applications range from $10,000.00 to $30,000.00," and "[t]he average costs for *prosecuting* the application before the patent office is from $10,000.00 to $20,000.00") (emphasis added) (footnotes omitted). In total, the average cost for the whole software patent process range from $20,000 to $50,000. *See id.* Furthermore, according to Kennard, prior art searches can cost the software developer anywhere from $2,000 to $20,000, if they go about the search in the most effective way. *See id.* "[H]owever, the costs can go far beyond this for prior art that is hard to find." *Id.*

114. *See* FTC Report, *supra* note 81, ch. 1, at 30; *see also* Kennard, *supra* note 98, at 336 ("[T]he average number of prior art references in the categories of U.S. patents, foreign patents, and non-patent references is very small. Therefore, the possibility of there being prior art that can have an [e]ffect on the validity and enforceability of the [sic] one or more of the claims of the patent at issue is relatively high."). Furthermore, these invalidating prior art references are more likely to be found during trial due to the amount of damages at stake. *See* Kennard, *supra* note 98, at 336 n.26.

115. *See* Cohen & Lemley, *supra* note 112, at 43. As Professors Cohen and Lemley explained:

> Because the vast majority of software innovation takes place outside traditional research institutions, many software improvements are recorded in ways that tend to elude the formal system of technical documentation followed in fields more closely linked to the scientific and technical establishment . . . . Frequently, the source code itself is never released at all. As a result, priority searches for software patents can be enormously difficult.
>
> Commentators, industry insiders, and the PTO itself have recognized that the lack of a comprehensive record of innovation in the software industry has important consequences for the patent prosecution process. The patent system presumes a finite, comprehensively indexed technical literature and relies on individual examiners to define, access, and search the relevant subliteratures. In the last several years, the PTO has taken measures to improve examiner access to nontraditional sources of software documentation, but the diffuse nature of the knowledge base and the lack of a comprehensive system for cataloguing [sic] and indexing software-related developments defy even the most knowledgeable and diligent examiner. It is just harder, maybe even impossible, for any one individual to find all relevant information, even in a perfect world. And since examiners work under incredible time constraints, particularly in the software-related units currently flooded with applications, they simply do not have time to find and to analyze what software prior art is scattered throughout the PTO classification system . . . . Thus, even as the number of issued software patents approaches twenty thousand per year, significant deficits in the PTO's ability to examine software patent applications remain unaddressed. As a result, software patents are more likely than other types of patents to receive a broader scope at the outset than some might say they deserve.

Id. at 43–44.

often encompass incremental innovations made by a subsequent software developer who is then found to be infringing.[116] This can be attributed to abstract software technology,[117] which, when patented, often consists of abstract claims.[118] "The distinguishing feature of an abstract patent claim is not that it covers a broad range of technologies, although that is often the case, but rather that it claims technologies unknown to the inventor."[119] Thus, the patent system creates a stronger monopoly of rights for software developers than is reasonable and leads to the prevalent problems of patent thickets and defensive patent uses in the software industry that make it substantially harder for subsequent software developers to innovate.[120]

These broad software patents are also often "questionable" as a result of insufficient prior art searches.[121] Proponents of patents argue that individuals and smaller organizations would benefit the most from patent protection.[122] But, contrary to that assertion, questionable software patents make market entry difficult for smaller software developers

---

116. *See id.* at 43–46 (explaining patent infringement and its related doctrines and how inadequate prior art disclosures combined "with the highly incremental character of software innovation" causes "a broad 'umbrella effect' for issued software patents").

117. *See supra* note 103.

118. *See* Bessen & Meurer, *supra* note 5, at 187 ("Although not all software patents contain abstract claims, the technology facilitates abstract claiming.").

119. *Id.* at 199.

120. *See supra* notes 85–90 and accompanying text; *see also* Bessen & Meurer, *supra* note 5, at 199–200. As Bessen and Meurer explained:

> There are two inter-related problems with such abstract claims. First, these claims reward patentees for inventions they do not invent. This means that the actual, future inventors face reduced incentives because they have to obtain a license from the patentee to develop or to commercialize their inventions. Clearly this counters the social benefit of the patent system. Second, it may be difficult to determine the boundaries of such claims and thus it may be difficult to provide notice, to conduct clearance searches, or to even determine the content of the prior art. The problem of mapping words to technology is difficult and it is made more difficult if the claims are not tethered to a specific device or to a specific physical or chemical process. Patent lawyers use the phrase "the embodiments of the invention" to describe the specific devices and processes disclosed in the patent document. Courts often interpret the meaning of the words in a claim in light of the specific embodiments of the invention . . . .
>
> . . . [T]he words in an abstract claim map to an uncertain set of technologies when they are not limited to distinct embodiments . . . . Sometimes, the progress of technology will render this mapping increasingly uncertain over time.
>
> Second, software patents may be particularly prone to strategic use of vague language by applicants to gain undeserved scope . . . . Although clever lawyers can use vague language with any technology, abstract technologies particularly lend themselves to such abuses because they are inherently described in abstract terms.

Id.

121. *See* FTC Report, *supra* note 81, ch. 3, at 53–55.

122. *See generally* Bessen & Meurer, *supra* note 5, at 165–86 (discussing whether small inventors benefit from the patent system).

"who lack the resources to challenge such patents."[123] These questionable patents also create disincentives to innovate for all software developers. As the FTC stated:

> A questionable patent can raise costs and prevent competition and innovation that otherwise would benefit consumers . . . . [M]any panelists in knowledge-based industries such as . . . software asserted that, because of questionable patents, they must steer their innovative efforts away from potentially productive areas, accede to possibly unjustified licensing terms, or enter into cross-licensing agreements that effectively "contract out" of the patent system.[124]

The FTC also found that firms "use questionable patents to extract high royalties or to threaten litigation," and that "a questionable patent that claims a single routine in a software program may be asserted to hold up production of the entire software program," which "can deter follow-on innovation and unjustifiably raise costs to businesses and, ultimately, to consumers."[125]

Moreover, software is not only incompatible with patent procedures, but also with patent legal standards. Software development operates incrementally based upon the success of older inventions .[126] As Professor Samuelson et al. noted:

> Classical intellectual property regimes do not protect this kind of innovation. Patent law requires an *inventive advance* over the prior art before it grants protection. Protecting incremental innovations in program behavior through patent law would thwart the economic goals of the patent system: to grant exclusive rights only when an innovator has made a *substantial* contribution to the art and *advanced competition to a new level*.[127]

This "inventive advance"[128] is a hallmark of patent law derived from the non-obviousness standard of § 103 of the Patent Act.[129] A person skilled in the art of software

123. *See* FTC Report, *supra* note 81, ch. 3, at 54.

124. FTC Report, *supra* note 81, ch. 4, at 1. Patent litigation can result in legal costs that number in the millions of dollars. *See* Bessen & Meurer, *supra* note 5, at 132.

125. FTC Report, *supra* note 81, at 7 (stating that questionable patents are used to extract high royalties through licenses which deters subsequent innovation, thus raising the cost of doing business and passing it on to consumers). *See also* Bessen & Meurer, *supra* note 5, at 199 (arguing that because software patents consist of abstract claims which contribute to the USPTO issuing questionable patents, "these claims reward patentees for inventions they do not invent" and that "this means that the actual, future inventors face *reduced* incentives because they have to obtain a license from the patentee to develop or to commercialize their inventions").

126. *See supra* notes 85–86 and accompanying text.

127. Samuelson et al., *supra* note 9, at 2346 (emphasis added).

128. *Id.*

129. *See* 35 U.S.C. § 103 (2006) (describing the "conditions for patentability; non-obvious subject matter"); Graham v. John Deere Co. of Kan. City, 383 U.S. 1, 14–15 (1966) ("Section 103, for the first time in our statute, provides a condition which exists in the law and has existed for more than 100 years, but only by reason of decisions of the courts. An invention which has been made, and which is new in the sense that the same thing has not been made before, may still not be patentable if the difference between the new thing and what was known before is not considered sufficiently great to warrant a patent. That has been expressed in a large variety of ways in decisions of the courts and in writings. Section 103 states this requirement in the title. It refers to the difference between the subject matter sought to be

development generally makes incremental or minor improvements over existing software technology.[130] Section 103 dictates that such an invention with only an incremental improvement is non-patentable because it would be "obvious" to someone skilled in the art of the invention.[131] However, anecdotal evidence suggests that "obvious" software patents are often issued despite the prohibition of such patents in § 103.[132] Thus, the incremental nature of software technology and development in the software industry is irreconcilable with patent law's non-obviousness standard.

Similarly, Professors Bessen and Meurer also discuss this issue, but in terms of "trivial inventions," which "are at best trivial improvements on existing knowledge; at worst, they are blatantly obvious."[133] According to Professors Bessen and Meurer, trivial software patents along with abstract software claims cause notice problems for subsequent software developers:

> It is possible that features of software technology make it particularly susceptible to the patenting of obvious ideas, especially given the legal doctrines of non-obviousness developed by the Federal Circuit. For one thing, the general purpose nature of software technology—again, because the technology is abstract, similar techniques can be used in a wide range of applications—means that techniques known in one realm might be applied in another, yet the documentary evidence that the Federal Circuit requires for a demonstration of obviousness might not be published [the non-patent prior art] . . . . Whatever the cause, the combination of large numbers of software patents that are both trivial and abstract produces significant problems of patent notice.[134]

In turn, notice problems contribute to increased software patent litigation,[135] which lead to decreased R&D investments.[136]

---

patented and the prior art, meaning what was known before as described in section 102. If this difference is such that the subject matter as a whole would have been obvious at the time to a person skilled in the art, then the subject matter cannot be patented.").

130. *See supra* notes 84–85 and accompanying text.

131. *See supra* notes 129; *cf.* KSR Int'l Co. v. Teleflex Inc., 550 U.S. 398, 419 (2007) ("In many fields it may be that there is little discussion of obvious techniques or combinations, and it often may be the case that market demand, rather than scientific literature, will drive design trends. Granting patent protection to advances that would occur in the ordinary course without real innovation retards progress . . . .").

132. *See, e.g.*, Arti K. Rai, John R. Allison & Bhaven N. Sampat, *Frontiers in Empirical Patent Law Scholarship: University Software Ownership and Litigation: A First Examination*, 87 N.C. L. Rev. 1519, 1520–21 (2009) ("Various scholars have quarreled with the alleged vagueness and undue breadth of software patent claims. Some have also suggested that, given the poor quality of prior art documentation and patent examiner training in the area of software, many issued software patents are likely to be obvious."); FTC Report, *supra* note 81, at 10; Richard Stallman, *The Anatomy of a Trivial Patent*, http://www.gnu.org/philosophy/trivial-patent.html (last visited Oct. 31, 2010) ("Programmers are well aware that many of the software patents cover laughably obvious ideas.").

133. *See* Bessen & Meurer, *supra* note 5, at 212.

134. *See id.* at 212–13.

135. *See id.* at 164 ("The evidence suggests . . . that the deterioration of the notice function might be the central factor fueling the growth in patent litigation.").

136. *See supra* note 98 and accompanying text.

Lastly, software further deviates from patent norms with regard to the disclosure requirement. Software patent proponents argue that the disclosure requirement for patents should drive progress and innovation in the software industry.[137] However, software patentees do not have to reveal source code in their disclosure to explain their invention.[138] Without meaningful disclosure, the extent to which the disclosed software invention is helpful or contributes to a subsequent inventor's innovation is unclear. Indeed, the anecdotal evidence signifies that these disclosures often do not contribute to innovation.[139] Furthermore, inadequate disclosures also contribute to the notice problems identified by Professors Bessen and Meurer.[140] One thing is certain from all of this: the software industry would benefit without patents.

V. SOFTWARE SHOULD BE INELIGIBLE FOR PATENT PROTECTION

A. Abstract Ideas and Algorithms Are Barred Under the Patent Act

Under a utilitarian theory, patents are rewards for software developers who produce inventive software, and in return, these rights should encourage further software development.[141] Instead, patents create disincentives for software developers who want to develop software. This effect is supported by ample statistical and anecdotal evidence, which shows that software patents reduce R&D funds, increase

137. *See supra* Part II.C.

138. Fonar Corp. v. Gen. Elec. Co., 107 F.3d 1543, 1549 (Fed. Cir. 1997) ("As a general rule, where software constitutes part of a best mode of carrying out an invention, description of such a best mode is satisfied by a disclosure of the functions of the software. This is because, normally, writing code for such software is within the skill of the art, not requiring undue experimentation, once its functions have been disclosed. It is well established that what is within the skill of the art need not be disclosed to satisfy the best mode requirement as long as that mode is described. Stating the functions of the best mode software satisfies that description test. We have so held previously and we so hold today. Thus, flow charts or source code listings are not a requirement for adequately disclosing the functions of software." (citation omitted)).

139. *See* FTC Report, *supra* note 81, ch. 2, at 7 n.47, ch. 4, at 25 nn.148–49.

140. *See* Bessen & Meurer, *supra* note 5, at 199–200. *See also* Robert M. Hunt, *Economics and the Design of Patent Systems*, 13 Mich. Telecomm. Tech. L. Rev. 457, 463–64 (2007) ("If patent law's disclosure requirements are not adequately enforced . . . . one might not be certain what the applicant has invented and how far his or her claims should extend. In these areas, some researchers and practitioners worry that applicants can obtain relatively broad patents even though they have not really started their R&D." (footnote omitted)); Burk & Lemley, *supra* note 112, at 1165–66 ("It is simply unrealistic to think that one of ordinary skill in the programming field can necessarily reconstruct a computer program given no more than the purpose the program is to perform. Programming is a highly technical and difficult art. Unfortunately, the Federal Circuit's peculiar direction in the software enablement cases has effectively nullified the disclosure requirement for software patents. And since source code is normally kept secret, software patentees generally disclose little or no detail about their programs to the public. Software patentees during the 1980s and early 1990s tended to write their patents in means-plus-function format in order to satisfy the changing dictates of the Federal Circuit's patentable subject matter rules. Lawyers writing patents in such a format have an incentive to describe their invention in the specification in terms that are as general as possible, since means-plus-function claim elements will be limited to the actual structure disclosed in the specification and equivalents thereof. As a result, there is no easy way to figure out what a software patent owner has built except to reverse engineer the program." (footnote omitted)).

141. *See supra* notes 63–68 and accompanying text.

likelihoods of patent infringement, and cause anti-competitive behavior from patent owners using defensive software patents.[142]

The Constitution authorizes Congress to enact laws which protect these intellectual property rights provided that they "promote the Progress of Science and useful Arts."[143] Yet patent law seems to be hindering progress in the software industry by creating obstacles to software development and innovation,[144] primarily because software, as an industry and technology, is unable to adjust to patent norms.[145] Therefore, as a policy matter, the software industry would be better served without the impediments that patents create.

But there is also a legal justification as to why software should not be patented. A basic principle under patent law is that "laws of nature, natural phenomena, and abstract ideas" *cannot* be patented.[146] Algorithms fall under this exception as well.[147] The reason for this abstract ideas and algorithms exception is that if we issue patents on these fundamental concepts, then we are effectively preempting all subsequent inventors from gaining access to that which belongs to the public domain. More specifically, when an invention involves a process that covers a law of nature, a natural phenomenon, an abstract idea, or an algorithm, the "claim is so abstract and sweeping as to cover both known and unknown uses" of the underlying process and end result.[148] Because of the nature of the technology, software claims clearly suffer from this sort of patenting problem.[149] In other words, if a process patent uses a law of nature, a natural phenomenon, an abstract idea, or an algorithm as its process to produce a particular result, then a future inventor who may someday derive a completely different kind of result from that process, or use the process in another

142. *See supra* Part III.

143. U.S. Const. art I, § 8, cl. 8.

144. *See supra* Part III.

145. *See supra* Part IV.

146. Diamond v. Diehr, 450 U.S. 175, 185 (1981).

147. *Id.* at 186.

148. *See, e.g.*, O'Reilly v. Morse, 56 U.S. 62, 113 (1854). In holding that Samuel Morse, inventor of the telegraph, could not patent a claim for electromagnetism associated with his telegraph, the Supreme Court explained that

> [i]f this claim can be maintained, it matters not by what process or machinery the result is accomplished. For aught that we now know some future inventor, in the onward march of science, may discover a mode of writing or printing at a distance by means of the electric or galvanic current, without using any part of the process or combination set forth in the plaintiff's specification. His invention may be less complicated—less liable to get out of order—less expensive in construction, and in its operation. But yet if it is covered by this patent the inventor could not use it, nor the public have the benefit of it without the permission of this patentee.

Id.

149. *See* Gottschalk v. Benson, 409 U.S. 64, 68 (1972).

fashion, is barred from doing so because of the patent claims covering that process.[150]

Accordingly, the Supreme Court has found certain software claims to fit within this "abstract idea" exception, beginning with its decision in *Benson*.[151] A preemption concern was the *Benson* court's primary rationale behind holding the computer program at issue unpatentable.[152] Prescient of the problems patenting software may create, the *Benson* court cautioned that:

> If these programs are to be patentable, considerable problems are raised which only committees of Congress can manage, for broad powers of investigation are needed, including hearings which canvass the wide variety of views which those operating in this field entertain. The technological problems tendered in the many briefs before us indicate to us that considered action by the Congress is needed.[153]

Unfortunately, Congress never intervened to determine software's patentability status, and instead, the courts had to resolve this issue.

More recently, after years of contention and the lower federal courts fashioning various (often vague) tests for statutory subject matter under the Patent Act,[154] the Supreme Court in *Bilski v. Kappos* was again presented with the question of whether software claims are directed to statutory subject matter.[155] Although the issues presented on appeal in *Bilski* were narrowly framed around business method patents, amici curiae asked the Court to also consider whether software patents were invalid.[156] The Supreme Court, however, declined to consider that issue.[157] As a result, there will be a continued debate among scholars and software developers as to whether

150. *See supra* note 118–21 and accompanying text.

151. *See Benson*, 409 U.S. at 64, 71–73 (reversing the Court of Customs and Patent Appeals' (the predecessor to the Federal Circuit) decision to sustain patent claims directed to a computer program and finding that if these claims were allowed, it would result in the patenting of an idea which is prohibited under patent law).

152. *Id.* at 71–72 ("It is conceded that one may not patent an idea . . . . The mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself. It may be that the patent laws should be extended to cover these programs, a policy matter to which we are not competent to speak.").

153. *See id.* at 73.

154. *See supra* Part II.B.

155. *See* Bilski v. Kappos, 130 S. Ct. 3218, 3227–28 (2010).

156. *See, e.g.*, Red Hat Amicus Curiae Brief, *supra* note 15 (arguing that software should *not* be patented); Brief for Business Software Alliance as Amicus Curiae Supporting Neither Party & Supporting Affirmance, Bilski v. Doll, 77 U.S.L.W. (U.S. June 1, 2009) (No. 08-964) (arguing that software should be patented).

157. *Bilski*, 130 S. Ct. at 3228 ("It is important to emphasize that the Court today is not commenting on the patentability of any particular invention, let alone holding that any of the . . . technologies from the Information Age should or should not receive patent protection.")

software should qualify as patentable subject matter under the Patent Act, and the validity of software patents will continue to be uncertain.[158]

When the Court one day reaches this specific issue, this note proposes that the Court should adopt a per se exception for software that would exclude it from statutory subject matter under the fundamental rationale that abstract ideas or algorithms are simply unpatentable.[159] Under this exception, the Court could find that software claims are directed to some abstraction that would effectively preempt matters that rightfully belong in the public domain in violation of established patent doctrine. The Court avoided such a broad holding in *Bilski v. Kappos* for all business methods and other purported processes.[160] However, with regard to software claims, it is clear that when boiled down to its basic components, software is really nothing more than a written expression of abstract ideas and algorithms.[161] As *Benson* acknowledged:

> A principle, in the abstract, is a fundamental truth; an original cause; a motive; these cannot be patented, as no one can claim in either of them an exclusive right. Phenomena of nature, though just discovered, mental processes, and abstract intellectual concepts are not patentable, as they are the basic tools of scientific and technological work.[162]

Although the Court upheld the validity of the machine-or-transformation test,[163] a per se exception barring software would obviate the need for the Court to craft another statutory subject test that must also answer the question of software's patentability. For a unique technology such as software, a per se exception is necessary

158. *See, e.g.*, David Worthington, *Supreme Court Strikes Down Bilski Patent Claim*, Software Dev. Times (June 28, 2010), http://www.sdtimes.com/link/34447; Wadhwa, *supra* note 85.

159. *See* Gottschalk v. Benson, 409 U.S. 64, 71–72 (1972) ("It is conceded that one may not patent an idea . . . . The mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below is affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself. It may be that the patent laws should be extended to cover these programs, a policy matter to which we are not competent to speak.").

160. *See Bilski*, 130 S. Ct. at 3231 ("[T]he Court once again declines to impose limitations on the Patent Act that are inconsistent with the Act's text. The [business method] patent application here can be rejected under our precedents on the unpatentability of abstract ideas. The Court, therefore, need not define further what constitutes a patentable 'process,' beyond pointing to the definition of that term provided in § 100(b) [of the Patent Act] and looking to the guideposts in *Benson*, *Flook*, and *Diehr*.").

161. Samuelson et al., *supra* note 9, at 2321 n.37 ("[P]rograms are built from information structures, such as algorithms and data structures . . . . An algorithm is a 'prescribed set of well-defined, unambiguous rules of processes for the solution of a problem in a finite number of steps'; data is 'a formalized representation of facts or concepts suitable for communication, interpretation, or processing by people or by automatic means'; a data structure is the structure of relationships among data items." (citation omitted)).

162. *Benson,* 409 U.S. at 67 (citation omitted).

163. *See Bilski*, 130 S. Ct. at 3227.

due to the numerous adverse effects that software patents cause[164] and their inherent abstract nature that makes them incompatible with patent norms.[165]

Moreover, it is not apparent whether the machine-or-transformation test would actually eliminate software claims, even though that seems to be the effect at the moment.[166] The first prong of the test requires that a claim be "tied to a particular machine."[167] The Federal Circuit in *Bilski*, however, did not explain this requirement any further because the machine issue was not before them, and the court could not answer "whether or when recitation of a computer suffices to tie a process claim to a particular machine."[168]

The Supreme Court, while upholding the machine-or-transformation test's validity, also did not give any further guidance as to the extent in which the test would reject software claims. Rather, the Court's decision in *Bilski v. Kappos* made the machine-or-transformation test even more unclear, while also creating further ambiguity as to what test the lower federal courts should use to determine whether a software claim is directed towards a valid process under the statutory subject matter requirement of § 101 of the Patent Act.[169]

In *Gutta*, the BPAI explained that a "general purpose computer," which most patent applicants tie their software claims to, would fail the first requirement of the *Bilski* case in the Federal Circuit:

> Process claims 1 and 7 recite "[a] computerized method performed by a data processor." Claim 1 additionally requires, "displaying the [calculated result] to [a] target user." These are the only limitations which could arguably be construed to tie the claimed process to a particular machine under the first prong of the machine-or-transformation test. This is the exact issue that the court in *Bilski* declined to decide. The court did, however, provide some guidance when it explained that the use of a specific machine must impose meaningful limits on the claim's scope to impart patent-eligibility.
>
> The recitation in the preamble of "[a] computerized method performed by a data processor" adds nothing more than a general purpose computer that is associated with the steps of the process in an unspecified manner. Such a field-of-use limitation is insufficient to render an otherwise ineligible process

164. *See supra* Part III.

165. *See supra* Part IV.

166. *See supra* note 14 and accompanying text (citing cases where the BPAI rejected software claims under the machine-or-transformation test).

167. *In re* Bilski, 545 F.3d 943, 961 (Fed. Cir. 2008).

168. *Id.* at 962.

169. *See Bilski*, 130 S. Ct. at 3227–28, 3231 ("This Court's precedents establish that the machine-or-transformation test is a useful and important clue, an investigative tool, for determining whether some claimed inventions are processes under § 101. The machine-or-transformation test is not the sole test for deciding whether an invention is a patent-eligible 'process . . . .' In disapproving an exclusive machine-or-transformation test, we by no means foreclose the Federal Circuit's development of other limiting criteria that further the purposes of the Patent Act and are not inconsistent with its text.").

> claim patent eligible. This recitation, therefore, fails to impose any meaningful limits on the claim's scope.[170]

If the Supreme Court in *Bilski v. Kappos* had elaborated further on the machine prong of the machine-or-transformation test, as the BPAI did in *Gutta*, then perhaps a per se exception for software would be unnecessary, as most software claims would consequently fail. However, the Court, without further elaboration, only held that the test was but one of a number of potential tools available to the Federal Circuit to limit invalid process claims.[171] Under the machine-or-transformation test as it exists now, a software claim could still potentially pass muster under the transformation prong of the test if the claim transforms data into a "visual depiction" of something physical.[172] While this visual depiction limit would seem to bring many abstract software claims outside the scope of statutory subject matter so that many claims are not patentable,[173] it is not entirely certain that it will bar *all* software claims because most software output some sort of physical display. In that case, certain software claims would still survive. Thus, a per se exception barring all software claims under § 101 of the Patent Act is necessary to avoid any uncertainties as to whether software can be patented. More importantly, it would also avoid any creative drafting of software claims that is intended to maneuver around whatever statutory subject matter test the Federal Circuit decides to adopt.

B. *Eliminating Software Patents Poses No Undue Hardships on the Software Industry*

Currently, software has three forms of substantial legal protection: copyright, patents, and trade secrets. Therefore, a per se exception eliminating patents from a software developer's arsenal would not leave them unprotected or with fewer incentives to develop software. Drawing a parallel to a time when software patents did not exist or had not yet begun to define the software industry is appropriate here. In that time, many important software innovations were created without the protection or incentive of patents.[174] Furthermore, in *Benson*, the Supreme Court

170. *Ex parte* Gutta, No. 2008-3000, 2009 WL 112393 (B.P.A.I. Jan. 15, 2009).

171. *See Bilski*, 130 S. Ct. at 3227.

172. *In re* Bilski, 545 F.3d 963 (Fed. Cir. 2008) ("So long as the claimed process is limited to a practical application of a fundamental principle to transform specific data, and the claim is limited to a visual depiction that represents specific physical objects or substances, there is no danger that the scope of the claim would wholly pre-empt all uses of the principle.").

173. *See, e.g.*, *Ex parte Gutta*, No. 2008-3000 at 6 ("The steps of process claims 1 and 7 also fail the second prong of the machine-or-transformation test because the data does not represent physical and tangible objects. Rather, the data represents information about user selection histories, an intangible.").

174. *See* Red Hat Amicus Curiae Brief, *supra* note 15 ("[M]ajor innovations and economic successes in the software industry occurred prior to the Federal Circuit's decisions in the mid-1990s encouraging software patents. Such enormously successful software products as Microsoft Word, Oracle Database, Lotus 1-2-3, the Unix operating system, and the GNU C compiler all date from the 1980s or earlier—well before the proliferation of software patents.").

noted that "'the creation of programs has undergone substantial and satisfactory growth in the absence of patent protection . . . .'"[175]

A prime example is Microsoft, which, in 1991, owned only eight patents.[176] By then, it was already "the world's largest computer-software company," projected to generate over one billion dollars in revenue that year.[177] If the software industry was able to grow before increasing patent protection, then the same could be said today, particularly considering the negative impact that software patents have had and continue to have on software development.[178]

Software will still receive the generous benefits of copyright protection, which arguably provides a more optimum balance of incentives and competitive considerations for the software industry. As Professors Bessen and Maskin noted: "The ideal patent policy limits 'knock-off' imitation, but allows developers who make similar, but potentially valuable complementary contributions. In this sense, copyright protection for software programs . . . may have achieved a better balance than patent protection." [179] This is because a software copyright protects the "expression" of an idea or algorithm (i.e., the software code), but does not restrict the dissemination of those concepts contained within the software.[180] In other words, copyright prevents freeloaders from copying the software, but allows another software developer to come along and use a substantially different expression (i.e., software code), to achieve the same result or to build upon that result.[181] Once a software patent is issued, however, those concepts are monopolized within the patent grant, and a subsequent software developer cannot create or use patented software which has any claims to

175. Gottschalk v. Benson, 409 U.S. 63, 72 (1972) (citing President's Comm'n on the Patent Sys., Report of the President's Commission on the Patent System, To Promote the Progress of . . . Useful Arts (1966)).

176. *See* Timothy B. Lee, *A Patent Lie*, N.Y. Times, June 9, 2007, http://www.nytimes.com/2007/06/09/opinion/09lee.html. *But see* Torsten Busse, *Software Floods the Patent Office*, Infoworld, Sept. 30, 1991, at 42 ("Microsoft Corp. has only nine patents to date.").

177. David Rensin, *Bill Gates: Soft Icon*, Playboy, Sept. 1991, at 134.

178. *See supra* Part III.

179. Bassen & Maskin, supra note 9, at 20.

180. *See* FTC Report, *supra* note 81, ch. 3, at 46 ("Copyright protects only the *expression* contained within a work,' not 'the underlying ideas expressed in that work." (quoting Roger E. Schechter & John R. Thomas, Intellectual Property: The Law of Copyrights, Patents, and Trademarks § 3.3, at 31–32 (2003))).

181. *See* Feist Publ'ns, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 349–50 (1991) ("The primary objective of copyright is not to reward the labor of authors, but 'to promote the Progress of Science and useful Arts.' To this end, copyright assures authors the right to their original expression, but encourages others to build freely upon the ideas and information conveyed by a work. This principle, known as the idea/expression or fact/expression dichotomy, applies to all works of authorship."); Cohen et al., *supra* note 6, at 327 (explaining the copyright infringement doctrine of "substantial similarity" and that a defendant is liable for copyright infringement if the defendant "engaged in actionable copying (i.e., copying in violation of § 106(1) [providing for a copyright holder's exclusive reproduction right]) by taking 'too much' of the plaintiff's work"). Thus, a software developer's expression must be substantially different from the original copyrighted software in order to avoid violating any of the exclusive rights under § 106 of the Copyright Act. *See* 17 U.S.C. § 106 (2006).

those ideas or algorithms without the patentee's authorization,[182] even if the subsequent developer expresses them in a completely unique manner. In this respect, copyright seems to promote progress in the software industry better than patents.

Furthermore, copyright may also have advantages over patents which would make patent protection for software superfluous in most cases. For instance, software technology is "fast-moving."[183] Some software developers, therefore, have only a limited time to take advantage of the market with their new invention before it becomes outdated. By the time a patent is granted for their software, new technology may have already overtaken the market.[184] The patent process simply cannot keep pace with how quickly some software develops. As the FTC found:

> Faster technology evolution and shorter product life cycles have increased the pressure on the PTO to reduce pendency times. As the U.S. House of Representatives Committee on Science/Subcommittee on Technology recognized: "In a growing number of industries—such as computer hardware and software . . . —the pace of advancement has begun to challenge the ability of the patent office to process applications in a time frame that is functionally useful to the inventor."[185]

With copyright, however, software is automatically protected as soon as the developer's code is written in a "tangible medium," provided that it meets the copyright standard of originality, which is a low standard to meet.[186] There is also no registration requirement for copyright protection which makes software development cheaper if patent costs are eliminated—an additional incentive that encourages software creation and progress.[187] Therefore, the software industry would suffer no harm from a per se software exclusion under patent law.

182. *See* 35 U.S.C. § 271 (2006) (providing for patent infringement); *In re Bilski*, 545 F.3d 953 (Fed. Cir. 2008) ("Patents, by definition, grant the power to exclude others from practicing that which the patent claims.").

183. *See* FTC Report, *supra* note 81, ch. 1, at 31.

184. *See* Kennard, *supra* note 98, at 332 ("One major liability is the time it takes to obtain a patent, which on average is from 18 months to 2 years. In that period of time, software in a fast changing area may eclipse the patented software invention.").

185. FTC Report, *supra* note 81, ch. 1, at 34.

186. *See* 17 U.S.C. § 102(a) (2006) ("Copyright protection subsists . . . in original works of authorship fixed in any medium of expression . . . from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device."); *see also Feist*, 499 U.S. at 345 ("Original, as the term is used in copyright, means only that the work was independently created by the author (as opposed to copied from other works), and that it possesses at least some minimal degree of creativity. To be sure, the requisite level of creativity is extremely low; even a slight amount will suffice. The vast majority of works make the grade quite easily, as they possess some creative spark, 'no matter how crude, humble or obvious' it might be. Originality does not signify novelty; a work may be original even though it closely resembles other works so long as the similarity is fortuitous, not the result of copying.").

187. *See* CONTU Final Report, *supra* note 21, at 16–17. In comparing copyright protection of software with patents, CONTU found that:

Additionally, copyright is an important tool that is used to facilitate the open source software movement.[188] The open source movement is becoming increasingly important in the software industry[189] and is said to be "an alternative means of fostering innovation" within it.[190] Signaling its ongoing importance, in 2000, "[t]he President's Information Technology Advisory Committee recommended that the federal government support open source software as a strategic national choice to sustain the U.S. lead in critical software development."[191] Accordingly, even if the software industry loses patent protection, innovation and progress will still thrive.

In light of the open source movement, the theory that patents are needed as economic incentives to compel programmers and organizations into creating and developing software is being further pushed to its limits. Indeed, the open source movement may even give credence to the idea that economic benefits can still be derived even though software code is made available to the public at no cost.[192] The

> In certain circumstances, proprietors may find patent protection more attractive than copyright, since it gives them the right not only to license and control the use of their patented devices or processes but also to prevent the use of such devices or processes when they are independently developed by third parties . . . . *The acquisition of a patent, however, is time-consuming and expensive*, primarily because a patentee's rights are great and the legal hurdles an applicant must overcome are high. A work must be useful, novel and non-obvious to those familiar with the state of the art in which the patent is sought. The applicant must prove these conditions to the satisfaction of the Patent and Trademark Office or, failing that, to the Court of Customs and Patent Appeals or the Supreme Court.

Id. (emphasis added).

188. *See, e.g.*, *GNU General Public License, version 2*, GNU Operating Sys., http://www.gnu.org/licenses/gpl-2.0.html (last visited Oct. 14, 2010); *Artistic License 2.0*, The Perl Found., http://www.perlfoundation.org/artistic_license_2_0 (last visited Oct. 14, 2010); *see also* Jonathan Zittrain, *Normative Principles for Evaluating Free and Proprietary Software*, 71 U. Chi. L. Rev. 265, 266 (2004) ("The legal forms of proprietary and free software production cannot coexist within a given piece of code. The proprietary form relies on the existence and enforcement of prevailing copyright law. In contrast, copylefted code asserts a thus far legally untested license pegged to copyright in order to establish the restriction that successor code must be licensed in precisely the same way, namely with its source code freely available.").

189. *See* FTC Report, *supra* note 81, ch. 3, at 48 ("Open source software has received considerable attention in recent years due to: (1) its rapid adoption, particularly by expert users and corporations; (2) significant capital investments in open source projects by corporations such as Hewlett Packard, IBM, and Sun Microsystems; and (3) the hailing of its collaborative nature of development by business and trade press as an important organizational innovation.").

190. *See id.* ch. 3, at 47–48 (noting that "[s]ome software representatives observed that copyrights or open source code policies facilitate the incremental and dynamic nature of software innovation").

191. Yochai Benkler, *Coase's Penguin, or, Linux and The Nature of the Firm*, 112 Yale L.J. 369, 371 (2002).

192. *See* Jacobsen v. Katzer, 535 F.3d 1373 (Fed. Cir. 2008). The Federal Circuit explained:

> The lack of money changing hands in open source licensing should not be presumed to mean that there is no economic consideration, however. There are substantial benefits, including economic benefits, to the creation and distribution of copyrighted works under public licenses that range far beyond traditional license royalties. For example, program creators may generate market share for their programs by providing certain components free of charge. Similarly, a programmer or company may increase its

fact that developers still have incentives to create software without patents contradicts their utilitarian justification and, once again, brings into question whether software patents are needed at all. In conclusion, the software industry does not need both copyright and patents as incentives to develop, and more importantly, the industry will likely grow even more without patent protection.

VI. CONCLUSION

Software is an important technology—our dependency on computers and the software that drives these machines is growing every day. The Founding Fathers were mindful of the important benefits that new technology would have on future generations and included the Progress Clause in the Constitution to ensure that laws would be enacted to promote new advancements. The Patent Act was created to achieve this objective, but has strained to accommodate the advances in software technology. Many in the software industry are particularly wary of the constraints that patent law imposes on innovation and competition, which is why software patents are so controversial. The debate as to whether software should be protected by patents will likely continue as long as there is evidence that patents have a harmful effect on the software industry. Because incentives to create software exist from alternative rights, and patents are restricting software development, the best policy to promote progress in the software industry would be to bar software from receiving patents.

> national or international reputation by incubating open source projects. Improvement to a product can come rapidly and free of charge from an expert not even known to the copyright holder.

Id. at 1379.