Patent Scope and Innovation in the Software Industry

Julie E. Cohen†
Mark A. Lemley‡

Table of Contents

† Associate Professor of Law, Georgetown University Law Center.

‡ Professor of Law, University of California at Berkeley, School of Law (Boalt Hall); of counsel, Fish & Richardson P.C. We would like to thank Fred Abbott, Erv Basinski, Dan Burk, Chris Byrne, Tom Cotter, Alan Durham, Richard Gruner, Rose Hagan, Paul Heald, Dennis Karjala, Ronald Mann, David McGowan, Peter Menell, Rob Merges, Mike Meurer, Tyler Ochoa, Margaret Jane Radin, Arti Rai, Pam Samuelson, Jay Thomas, Polk Wagner, David Welkowitz, participants in the 27th Annual Telecommunications Policy Research Conference, participants in faculty workshops at the Boston University School of Law and Whittier Law School for their comments on earlier versions, and Mitzi Chang and Elizabeth Monkus for research assistance. Need we mention that the ideas and mistakes contained herein are ours alone, and are not attributable to anyone else?

Patent Scope and Innovation in the Software Industry

Julie E. Cohen
Mark A. Lemley

Software patents have received a great deal of attention in the academic literature. Unfortunately, most of that attention has been devoted to the problem of whether software is or should be patentable subject matter. With roughly eighty thousand software patents already issued, and the Federal Circuit endorsing patentability without qualification, those questions are for the history books. The more pressing questions now concern the scope to be accorded software patents. In this Article, we examine the implications of some traditional patent law doctrines for innovation in the software industry. We argue that patent law needs some refinement if it is to promote rather than impede the growth of this new market, which is characterized by rapid sequential innovation, reuse and re-combination of components, and strong network effects that privilege interoperable components and products. In particular, we argue for two sorts of new rules in software patent cases.

First, we advocate a limited right to reverse engineer patented computer programs in order to gain access to and study those programs and to duplicate their unprotected elements. Such a right is firmly established in copyright law, and seems unexceptional as a policy matter even in patent law. But because patent law contains no fair use or reverse engineering exemption, patentees could use the grant of rights covering a single component of a complex program to prevent any "making" or "using" of the program as a whole, including those temporary uses needed in reverse engineering. While patent law does contain doctrines of "experimental use" and "exhaustion," it is not clear that those doctrines will protect legitimate reverse engineering efforts. We suggest that if these doctrines cannot be read broadly enough to establish such a right, Congress should create a limited right to reverse engineer software containing patented components for research purposes.

Second, we argue that in light of the special nature of innovation within the software industry, courts should apply the doctrine of equivalents narrowly in infringement cases. The doctrine of equivalents allows a finding of

infringement even when the accused product does not literally satisfy each element of the patent, if there is substantial equivalence as to each element. The test of equivalence is the known interchangeability of claimed and accused elements at the time of (alleged) infringement. A number of factors unique to software and the software industry—a culture of reuse and incremental improvement, a lack of reliance on systems of formal documentation used in other fields, the short effective life of software innovations, and the inherent plasticity of code—severely complicate post hoc assessments of the "known interchangeability" of software elements. A standard for equivalence of code elements that ignores these factors risks stifling legitimate, successful efforts to design around existing software patents. To avoid this danger, courts should construe software claims narrowly, and should refuse a finding of equivalence if the accused element is "interchangeable" with prior art that should have narrowed the original patent, or if the accused improvement is too many generations removed from the original invention.

Introduction

Software patents have received a great deal of attention in the academic literature. Unfortunately, most of that attention has been devoted to the problem of whether software is or should be patentable subject matter. With some eighty thousand software patents already issued,[1] the Federal Circuit endorsing patentability without qualification,[2] and the Supreme Court assiduously avoiding the question, software patentability is a matter for the history books. The more pressing questions now concern the criteria for issuance and the scope to be accorded issued software patents. And while public attention of late has been captured by so-called Internet business method patents, the overwhelming majority of such patents are in fact patents for software.[3] Thus, determining the scope of software patents will take us a long way towards determining what to do in practice with Internet business method patents as well.

As Part I of this Article describes, with software patents now being issued in large numbers, the patent system plays a newly prominent role in

---

1. *Infra* notes 33-38 and accompanying text.

2. *Infra* Part I.A.

3. Indeed, this was true of the prototypical business method patent, the one at issue in *State Street Bank & Trust v. Signature Financial Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998). In *State Street*, the Federal Circuit eliminated the long-standing rule against patenting non-technological "business methods." *See id.* The invention deemed patentable was a hub-and-spoke method of mutual fund accounting implemented in software. That ruling led to a host of patents and patent applications on various business ideas, many of which also are implemented in software and relate to the Internet and electronic commerce. *See, e.g.*, 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 (1999); Philip E. Ross, *Patently Absurd: Technology and Gamesmanship Have Overwhelmed the U.S. Patent Office. How to Fix It?*, Forbes, May 29, 2000.

shaping the development of the software industry. The consequences of this shift are worth examining more closely. Institutional mechanisms for encouraging innovation are a crucial determinant of the rate and nature of "progress" in technical fields.[4] Generally speaking, both economic theory and practical experience suggest that the availability of patents for software promotes innovation by supplying (additional) incentives to inventors.[5] Yet it is also possible that the patent system may constrain innovation if it draws protection too broadly.

Part I notes a convergence between the Patent and Trademark Office's [PTO] relatively unconstrained practice of issuing software patents and a strand of the theoretical literature which suggests that the optimal patent scope is broad. In the balance of the Article, we consider whether that result is the right one for the software industry. In particular, we examine the implications for software innovation of some traditional patent law doctrines affecting patent scope. We conclude that broad scope is not optimal, and that patent law needs refinement if it is to promote rather than impede the growth of this industry, which is characterized by rapid sequential innovation, reuse and recombination of components, and strong network

---

4. For examinations of the variety of institutional mechanisms available, see generally Brett Frischmann, *Innovation and Institutions: Rethinking the Economics of U.S. Science and Technology Policy*, 24 Vt. L. Rev. 347 (2000); Robert P. Merges, *Intellectual Property Rights and Collective Rights Organizations: Institutions Facilitating Transactions in Intellectual Property Rights*, 84 Calif. L. Rev. 1293 (1996); Arti Kaur Rai, *Regulating Scientific Research: Intellectual Property Rights and the Norms of Science*, 94 Nw. U. L. Rev. 77 (1999).

5. On the "reward theory" of patent protection, see The Subcomm. on Patents, Trademarks, and Copyrights of the Senate Comm. on the Judiciary, 85th Cong., An Economic Review of the Patent System (Comm. Print 1958). The extent to which the patent system is actually necessary to induce innovation that would not otherwise occur is an unanswered, and perhaps unanswerable, empirical question. *See generally id*.; George L. Priest, *What Economists Can Tell Lawyers About Intellectual Property*, 8 Res. L. & Econ. 19 (1986); *cf.* A. Samuel Oddi, *Beyond Obviousness: Invention Protection in the Twenty-First Century*, 38 Am. U. L. Rev. 1097 (1989) (arguing that patents should be issued only for major innovations). *But see* Arnold Plant, *The Economic Theory Concerning Patents for Inventions*, 1 Economica 30 (1934) (arguing that the availability of patent protection may yield supraoptimal levels of invention, at the expense of other socially valuable activity).

The bewildering variety of software innovations generated in the years before software was considered patentable suggests that for software, at least, patent protection may not be as necessary as the reward theory assumes. The question is complicated, however, by the availability of copyright protection for software during that period, and by uncertainty over both the scope of copyright protection and the degree of overlap between the copyright and patent models of protection. For discussion of that overlap, see, for example, Julie E. Cohen, *Reverse Engineering and the Rise of Electronic Vigilantism: Intellectual Property Implications of "Lock-Out" Programs*, 68 S. Cal. L. Rev. 1091 (1995); Dennis S. Karjala, *The Relative Roles of Patent and Copyright in the Protection of Computer Programs*, 17 J. Marshall J. Computer & Info. L. 41 (1998); A. Samuel Oddi, *An Uneasier Case for Copyright Than for Patent Protection of Computer Programs*, 72 Neb. L. Rev. 351 (1993); J.H. Reichman, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Colum. L. Rev. 2432 (1994); Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 Colum. L. Rev. 2308 (1994).

effects that privilege interoperable components and products. Accordingly, we argue for two sorts of rules in software patent cases.

In Part II, we advocate a limited right to reverse engineer patented computer programs to permit study of those programs and duplication of their unprotected elements.[6] Such a right is firmly established in copyright law, and seems unexceptional as a policy matter even in patent law. But because patent law contains no fair use or reverse engineering exemption, patentees could use the grant of rights covering a single component of a complex program to prevent any "making" or "using" of the program as a whole, including those temporary uses required for reverse engineering. Indeed, the *Sony v. Connectix* and *Sony v. Bleem* cases pending in the Ninth Circuit reflect an effort by a patent and copyright owner to do just that.[7] While patent law does contain doctrines of "experimental use" and "exhaustion," it is not clear that those doctrines will protect legitimate reverse engineering efforts. We suggest that if these doctrines cannot be read broadly enough to establish such a right, Congress should create a limited right to reverse engineer software containing patented components for research purposes.

In Part III, we argue that in light of the special nature of innovation within the software industry, courts adjudicating software cases should use caution to avoid applying the doctrine of equivalents too broadly. The doctrine of equivalents allows a finding of infringement even when the accused product does not literally satisfy each element of the patent, as long as there is substantial equivalence as to each element.[8] One test of equivalence is the "known interchangeability" of the claimed and accused elements at the time of alleged infringement. However, several factors unique to software and the software industry complicate post hoc assessments of "known interchangeability." The software industry is characterized by a culture of reuse and incremental improvement, a lack of reliance on systems of formal documentation used in other technical fields, the short effective life of software innovations, and the inherent plasticity of microcode. A standard for equivalence of code elements that ignores these factors risks stifling legitimate efforts to design around existing software patents. To avoid this danger, courts should beware of construing software claims too broadly, and should refuse a finding of equivalence if the accused element is "interchangeable" with prior art that should have

---

6. "Reverse engineering" refers to the process of working backwards from a finished product to discover how it was made. For discussion of the unique technical considerations that attend the reverse engineering of software, see *infra* notes 63-65 and accompanying text.

7. *See infra* notes 72-73 (discussing the *Sony* cases).

8. The scope of a patent is defined by its claims, which set out each element of the invention. Each element of the patent claim must be present in the accused device in order to find literal infringement. London v. Carson Pirie Scott & Co., 946 F.2d 1534, 1538-39 (Fed. Cir. 1991).

narrowed the original patent, or if the accused improvement is too many generations removed from the original invention.

Parts II and III of our paper are connected by a single common theme: a focus on the process of improvement and sequential innovation as it actually occurs in the software industry. We begin with reverse engineering, despite its ontological status as a "defense" in intellectual property litigation, because that is where many improvers begin. We then discuss what improvers do with the information they obtain and how those improvements are treated in the patent infringement context. We think this industry focus is central to a nuanced and pro-competitive application of patent law. Too often courts and commentators have focused narrowly on one doctrinal issue to the exclusion of others that interact with it.[9]

Some might object that our suggestions are "new" rules for the software industry that have no place within a patent system that is generalist by design. This issue, however, is largely a question of semantics. Resolution of patent disputes requires reference to the state of knowledge and the level of ordinary skill in the particular art under consideration.[10] Some industry-specific variation in the application of general legal rules is both inevitable and, we believe, appropriate. Further, our proposals are designed to restore parity between software patents and other sorts of patents, by giving software engineers the same sorts of rights and expectations that exist in other industries. We do not intend to propose a sui generis law of software patents. Rather, we think it is both possible and desirable to interpret existing law to achieve the results we suggest.

I
Software Patents: History, Practice, and Theory[11]

Software patents have a convoluted history. Within the legal system, the past three decades have witnessed an about-face on the question of software's eligibility for patent protection. As we recount in Part I.A, software's status as patentable subject matter was first doubted, then grudgingly admitted, and finally embraced. However, there has been considerable divergence between the "law on the books" and the law in action; in fact, approval of software patent applications was routine practice even before the courts recognized it.

Part I.B argues that the patentability debate has become a costly distraction from more practical, and increasingly pressing, questions about

9. Thus, to take just one example, the long debate about whether software was or should be patentable subject matter obscured the host of other legal issues that affect the validity and legal and competitive effects of software patents. *Infra* notes 176-185 and accompanying text (discussing the problem).

10. *E.g.*, 35 U.S.C. §§ 102, 103 (1994) (rules dependent on the level of skill in the industry).

11. The reader familiar with the law of software patents may wish to skip directly to Part II.

how the patent system should treat software. In Part I.C, we describe one strand of theoretical literature which antedates software patents, and which concludes that broad patents are economically optimal. In the balance of this Article, we conclude that as a result of certain characteristics of software and of research and development patterns within the software industry, issued software patents may enjoy very broad scope. The rapid rise of software patents thus affords an opportunity to test an important theoretical model, and to consider whether it is the right one for this industry. For a variety of reasons discussed in Parts II and III, we contend that it is not, and that courts should be careful to restrict the scope of software patents so that innovation will not suffer.

A. History: The Section 101 Patentability Debate

Today, it seems fairly settled that software-related inventions fall within the class of innovations described in section 101 of the Patent Act as eligible for patent protection. Thirty years ago, though, that conclusion was by no means foregone. Although the statute authorizes the patenting of any new and useful process or machine,[12] long-standing judicially developed doctrines prohibited patent protection for mathematical formulae and mental processes. The courts held that "processes" describing existing natural laws (whether as basic as $2 + 2 = 4$ or as complex as $E = mc^2$) or reciting steps performable by the human mind do not fall within the category of "useful arts."[13] Mathematical algorithms (not just formulae) were declared non-patentable subject matter in an early Supreme Court case, *Gottschalk v. Benson*.[14] Throughout the 1970s, courts generally rejected software patent applications on the grounds that software was really just a concatenation of unpatentable algorithms.[15]

---

12. 35 U.S.C. § 101 (1994) ("Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter . . . may obtain a patent therefor . . . .").

13. *E.g.*, Funk Bros. Seed Co. v. Kalo Inoculant Co., 333 U.S. 127, 130 (1947) ("[P]atents cannot issue for the discovery of the phenomena of nature . . . . [These] are part of the storehouse of knowledge of all men." (citation omitted)); *In re* Shao Wen Yuan, 188 F.2d 377, 380 (C.C.P.A. 1951); Halliburton Oil Well Cementing Co. v. Walker, 146 F.2d 817, 821 (9th Cir. 1944*), rev'd on other grounds*, 329 U.S. 1 (1946); Don Lee, Inc. v. Walker, 61 F.2d 58, 67 (9th Cir. 1932). *See generally* 1 Donald S. Chisum, Chisum on Patents § 1.03 (2000) (discussing the scope and boundaries of the statutory class of processes). The Patent Clause of the Constitution authorizes the grant of exclusive rights "to promote the Progress of . . . useful Arts." U.S. Const. art. I, § 8, cl. 8. The term "useful arts" has been construed to encompass "the realm of technological and industrial improvements." Pamela Samuelson, Benson *Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 Emory L.J. 1025, 1033 n.24 (1990); *see also id.* at 1112; 1 Chisum, *supra*, at § 1.01. As Professor Samuelson details, however, no coherent, satisfactory explanation or model has been offered for the exclusion of mathematical formulae and mental processes. *See* Samuelson, *supra*, at 1036 n.34.

14. 409 U.S. 63 (1972).

15. This history is well traced in Samuelson, *supra* note 13.

With *Benson* apparently precluding the patenting of "pure" software, patent applicants in the 1970s shifted their focus to patenting mechanical devices and processes that happened to include computer programs. The prototypical application in this category was for a "new" machine or process in a familiar art, in which the only point of novelty was the use of a computer program to run the machine or implement the process. Six years after *Benson*, in *Parker v. Flook*,[16] the Court rejected an attempt to patent a computerized method for continuously recalculating the "alarm limit" during a chemical conversion process. The *Flook* Court noted that the only novel feature of the invention was a computer program, and that the program itself was not patentable subject matter.[17]

Three years later, however, the Court changed its view. In *Diamond v. Diehr*,[18] it held that a process for continuously monitoring the temperature inside a synthetic rubber mold, using a computer and the well-known Arrhenius equation for measuring cure time as a function of temperature and other variables, was patentable subject matter. Central to the Court's decision was that the inventor did not claim all rights to future uses of the Arrhenius equation but only to the particular application that he had invented in the context of an "otherwise statutory" industrial process.[19] Despite this fact, and the Court's language insisting on significant "post-solution activity" outside the computer program,[20] *Diehr* seems difficult to distinguish from *Flook*.

The *Diehr* decision and its appellate progeny created what might be termed "the doctrine of the magic words." Under this approach, software was patentable subject matter, but only if the applicant recited the magic words and pretended that she was patenting something else entirely. During the 1980s and early 1990s, knowledgeable patent attorneys did exactly that, claiming software inventions as hardware devices, pizza ovens, and other "machines." As developed by the PTO and the Federal Circuit prior to 1994, the "otherwise statutory process or apparatus" limitation was not much of a limit at all.[21] Nearly any physical element or step would suffice to render statutory a claim that recited a mathematical or "mental process" algorithm, even if the physical element or step was well known or an industry standard and the mathematical algorithm was the only novel component of the invention.

---

16. 437 U.S. 584 (1978).

17. *Id.* at 589-91. The Court reasoned that if it ignored the mathematical algorithm the applicant had developed for updating the alarm limit, the claimed invention contained nothing new or inventive.

18. 450 U.S. 175 (1981).

19. *Id.* at 187.

20. *Id.* at 215.

21. Although derived from the Court's opinion in *Diehr*, this test became known as the *Freeman-Walter-Abele* test after the three appellate cases that elaborated it in greater detail. *In re* Freeman, 573 F.2d 1237 (C.C.P.A. 1978); *In re* Walter, 618 F.2d 758 (C.C.P.A. 1980); *In re* Abele, 684 F.2d 902 (C.C.P.A. 1982).

In 1994, the en banc Federal Circuit decided *In re Alappat*, opening a new era in software patent protection.[22] The decision established that the "otherwise statutory process or apparatus" requirement may be satisfied by the simple expedient of drafting claims to include a general purpose computer or standard hardware or memory element that would be necessary for any useful application of the algorithm. The *Alappat* court reasoned that "a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software."[23] Accordingly, the court ruled, it need not even perform the inquiry required by the *Freeman-Walter-Abele* line of cases. After *Alappat*, companies that wanted to patent software no longer needed to pretend they were patenting something else. They needed only to define their claims in terms of a computer program implemented in a machine.

The reasoning of *Alappat*, however, did not appear to encompass claims reading on computer programs themselves, as opposed to programs implemented in a machine or system. That obstacle to computer-related patent claims fell in 1995, when IBM appealed the PTO's rejection of a claim to "computer programs embodied in a tangible medium, such as floppy diskettes" to the Federal Circuit.[24] While the appeal was pending, the PTO decided not to oppose the claim. Shortly thereafter, it issued new examining guidelines for software patents that directed examiners to approve such claims.[25]

The remaining legal barriers to patenting "pure" software dissolved completely in 1998 when the Federal Circuit decided *State Street Bank & Trust v. Signature Financial Group*.[26] There, the court reversed a district court's rejection of a patent for a software-implemented financial system that automatically calculated and allocated profits from a joint stock account. The court concluded that the *Freeman-Walter-Abele* test "has little, if any, applicability to determining the presence of statutory subject matter."[27] Instead, it reasoned, even physical structure was unnecessary, so long as a process or idea was useful:

---

22. 33 F.3d 1526 (Fed. Cir. 1994) (en banc).

23. *Id.* at 1545. As a philosophical matter, this approach is troubling. As the dissent explained, "[w]hether or not subject matter is a 'new machine' within § 101 is precisely the same question as whether or not the subject matter satisfies the § 101 analysis . . . . [A] player piano playing Chopin's scales does not become a 'new machine' when it spins a roll to play Brahms' lullaby." *Id.* at 1566-67 (Archer, C.J., concurring in part and dissenting in part) (citations omitted). On the other hand, if the "machine" in question consists of the hardware combined with the software, the combination is certainly new. *Cf.* Alan L. Durham, *Useful Arts in the Information Age*, 1999 B.Y.U. L. Rev. 1419, 1519-20 (discussing the "new machine" approach of *Alappat*).

24. *In re* Beauregard, 53 F.3d 1583, 1584 (Fed. Cir. 1995).

25. United States Patent and Trademark Office, Examination Guidelines for Computer-Implemented Inventions, 61 Fed. Reg. 7478, 7479-80 (Jan. 1996).

26. 149 F.3d 1368 (Fed. Cir. 1998), *cert. Denied*, 525 U.S. 1093 (1999).

27. *Id.* at 1374.

> Today, we hold that the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces "a useful, concrete and tangible result"—a final share price momentarily fixed for recording and reporting purposes and even accepted and relied upon by regulatory authorities and in subsequent trades.[28]

The Federal Circuit affirmed this reasoning in *AT&T v. Excel Communications*.[29] There, the court upheld as patentable subject matter claims to a method for "generating a message record for an interexchange call" and recording to whom the call should be billed.[30] The court applied *State Street*'s "useful, concrete and tangible result" test and concluded that the generation of billing records was clearly useful.[31] Noting that physical transformation is only one of several possible ways to bring about a useful result, the court specifically rejected the argument that a patentable software claim must have physical structure associated with it.[32]

The end result of this history (and more than a quarter century of debate) is to leave the question of patentable subject matter very much where it would have been if *Benson* had come out the other way. As we will show, however, the protracted debate has nonetheless produced significant, though unintended, consequences for the patent system.

B. Practice: Anything Goes?

One might suppose that as a result of the long debate over software's eligibility for patent protection, software patents have only recently begun to issue in large numbers. Nothing could be further from the truth. Gradually, behind the scenes, and without the participation of the Supreme Court or even the Federal Circuit, software inventions of all types have been patented for some time. Close to one hundred thousand software or software-related patents are now in force in the United States, and several thousand more are being issued every year.[33] Numerous patents issued in

---

28. *Id.* at 1373.

29. 172 F.3d 1352 (Fed. Cir. 1999). On remand, the district court held the patent invalid under § 102. AT&T Corp. v. Excel Communications, 52 U.S.P.Q.2d 1865 (D. Del. 1999).

30. 172 F.3d at 1354.

31. *Id.* at 1361.

32. *Id.*

33. How many software patents exist depends in part on how one defines a software patent. Based on trends though mid-1998, Greg Aharonian projected that there would be over eighty thousand software patents in force as of early 2000, approximately forty thousand of which were issued by the end of 1995. *Internet Patent News Service*, at *http://swpat.ffii.org/penmi/bmwi-20000518/aharonian/stat-1998.txt* (visited June 16, 2000). John Allison and Mark Lemley estimate that during a two-year period in the late 1990s, the PTO issued approximately eighteen thousand software patents. John R. Allison & Mark A. Lemley, *Who's Patenting What? An Empirical Exploration of Patent Prosecution*, 53 Vand. L. Rev. (forthcoming 2000). These statistics suggest that the total number of

the 1980s and early 1990s cover pure data structures,[34] methods for performing calculations in a data processor,[35] data compression algorithms,[36] and software-based encryption algorithms,[37] despite the then-questionable statutory nature of such claims. Now, after *State Street* and *AT&T*, patents are being issued for software without any limitation as to tangible form, and for "propagated signals"—in effect, "signals" claims directed to "a manufactured transient phenomenon, such as an electrical, optical, or acoustical signal."[38] Like it or not, software patents are here to stay.

The confused judicial history of software patents has had important consequences for the present day, however. By focusing attention on the patentable subject matter debate, and giving at least lip service to the idea that software per se was unpatentable well into the 1990s, the court decisions we have discussed created a climate in which the actual patenting of software was largely ignored. As a result, the PTO only recently has begun to grapple with the difficult problems of identifying, cataloging, and searching for software prior art. In the meantime, tens of thousands of software patents have passed through the system.

For a variety of reasons, it is reasonable to think that these software patents have not been subject to the detailed examination for novelty and nonobviousness that they require. First, because software was not thought patentable on its own until recently, the PTO has only recently taken steps to hire patent examiners qualified in computer software or related fields.[39] During the 1980s and the early part of the 1990s, the flood of software patent applications was handled largely by people operating outside their area of expertise. Abundant evidence indicates that the PTO has issued software

---

existing software patents is no less than fifty thousand, and probably much higher. For earlier estimates, see Simson L. Garfinkel, *Patently Absurd*, Wired, July 1994, at 104, 106 (stating that over twelve thousand software patents had been issued by the end of 1993), and John T. Soma & B.F. Smith, *Software Trends: Who's Getting How Many of What? 1978 to 1987*, 71 J. Pat. & Trademark Off. Soc'y 415, 419-21, 428-32 (1989).

34. *E.g.*, U.S. Patent No. 5,488,717 (issued Jan. 30, 1996); U.S. Patent No. 5,414,701 (issued May 9, 1995).

35. *E.g.*, U.S. Patent No. 5,386,375 (issued Jan. 31, 1995).

36. *E.g.*, U.S. Patent No. 5,051,745 (issued Sept. 24, 1991).

37. *E.g.*, U.S. Patent No. 5,530,752 (issued June 25, 1996); U.S. Patent No. 4,405,829 (issued Sept. 20, 1983).

38. Jeffrey R. Keuster et al., *A New Frontier in Patents: Patent Claims to Propagated Signals*, 17 J. Marshall J. Comp. & Info. L. 75, 75 (1998) (discussing propagated signal claims); Gregory A. Stobbs, *Patenting Propagated Data Signals: What Hath God Wrought?*, IEEE Communications, July 2000, at 98 (same). Keith Witek offers an exhaustive guide to patenting computer programs and algorithms in a number of different forms, along with some analysis of the advantages and disadvantages of each, in Keith E. Witek, *Developing a Comprehensive Software Claim Drafting Strategy for U.S. Software Patents*, 11 Berkeley Tech. L.J. 363 (1996).

39. *See* Scott Thurm, *A Flood of Web Patents Stirs Dispute Over Tactics*, Wall St. J., Oct. 9, 1998, at B1 (noting that the PTO did not hire its first examiner with a degree in computer science until 1995). Indeed, until recently computer scientists were not even eligible to sit for the patent bar. *See* Cohen, *supra* note 5, at 1176.

patents on a number of applications that did not meet the standard tests of novelty and nonobviousness.[40] Second, for similar reasons, the PTO's classification system historically has not been equipped to handle software patents. As a result, software patents tended to be classified according to the field in which the software will ultimately be used (say, pizza ovens), rather than according to the nature of the software invention.[41] This in turn makes it much harder for examiners to find relevant prior art. Finally, prior art in this particular industry may simply be difficult or, in some cases, impossible to find because of the nature of the software business. Unlike inventions in more established engineering fields, most software inventions are not described in published journals. Software innovations exist in the source code of commercial products and services that are available to customers. This source code is hard to catalog or search for ideas.[42]

Commentators similarly have tended to neglect the non-subject matter issues associated with software patents. While there is a voluminous literature on whether software is (or should be) patentable subject matter,[43]

---

40. For anecdotes discussing some of the more extreme examples, see Garfinkel, *supra* note 33, at 104; Merges, *supra* note 3, at 588-91.

41. *See* Mark A. Lemley et al., Software and Internet Law 332 (2000).

42. As Julie Cohen has previously explained:

> [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. In the case of the multimedia data retrieval patent granted to Compton's New Media, industry criticism prompted the PTO to reexamine the patent and ultimately to reject it because it did not represent a novel and nonobvious advance over existing technology. However, it would be inefficient, and probably impracticable, to reexamine every computer program-related patent, and the PTO is unlikely to do so.

Cohen, *supra* note 5, at 1179 (footnotes omitted).

43. *See generally, e.g.*, Gregory A. Stobbs, Software Patents (1995); David S. Benyacar, *Mathematical Algorithm Patentability: Understanding the Confusion*, 19 Rutgers Computer & Tech. L.J. 129 (1993); Donald S. Chisum, *The Patentability of Algorithms*, 47 U. Pitt. L. Rev. 959 (1986); Irah H. Donner & J. Randall Beckers, *Throwing Out Baby Benson with the Bath Water: Proposing a New Test for Determining Statutory Subject Matter*, 33 Jurimetrics J. 247 (1993); Lee Hollaar, *Justice Douglas Was Right: The Need For Congressional Action On Software Patents*, 24 AIPLA Q.J. 283 (1996); Allen Newell, *The Models Are Broken, The Models Are Broken!*, 47 U. Pitt. L. Rev. 1023 (1986); Oddi, *supra* note 5; Samuelson, *supra* note 13; Samuelson et al., *supra* note 5; Richard H. Stern, *Tales from the Algorithm War:* Benson *to* Iwahashi, *It's Déjà Vu All Over Again*, 18 AIPLA Q.J. 371 (1991); Jur Strobos, *Stalking the Elusive Patentable Software: Are There Still* Diehr *or Was It Just a* Flook*?*, 6 Harv. J.L. & Tech. 363 (1993); John Swinson, *Copyright or Patent or Both: An Algorithmic Approach to Computer Software Protection*, 5 Harv. J.L. & Tech. 145 (1991); Jonathan N. Geld, Note, *General Does Not Mean Generic—Shedding Light on* In re Alappat, 4 Tex. Intell. Prop. L.J. 71 (1995); Maximilian R. Peterson, Note, *Now You See It, Now You Don't: Was It a Patentable Machine or an Unpatentable "Algorithm"? On Principle and Expediency in Current Patent Law Doctrines Relating to Computer-Related Inventions*, 64 Geo. Wash. L. Rev. 90 (1995). For a more recent approach focusing on constitutionality, but still in the context of patentable

there is much less discussion of other patent validity issues. Only in the latter part of the 1990s, as the Federal Circuit began to decide obviousness, enablement, and best mode cases involving software, did we start to see any significant discussion of these issues.[44]

Even less attention has been paid to questions of patent infringement and defenses to infringement claims. Despite what is now a large body of case law involving infringement of software patents, there is almost no academic treatment of the problem.[45] Only recently have commentators begun to discuss potential defenses to software patent infringement suits.[46] Our goal in the balance of this Article is to address these problems in an integrated way, with an eye towards the particular characteristics of the software industry.

C. Theory: Software Patents and the "Prospect" Theory of Patent Scope

The rapid introduction of large numbers of software patents into the patent system means that within a relatively short time, the background conditions for software innovation have been substantially reconfigured. Our analysis in Parts II and III suggests that because of the distinctive characteristics of software, these patents also may be accorded unprecedented breadth. In economic terms, this regime for software patents would resemble that outlined by Edmund Kitch in 1977, years before the question of software patentability became a pressing one.[47] The case of software patents thus offers a unique opportunity to assess the utility of an influential theoretical model.

Kitch based his "prospect" theory on an analogy to nineteenth-century mining claims, which reserved for first-comers all rights to explore the described terrain. Under the prospect theory of patent scope, issued patents would operate as broad reservations of rights in the technical landscape. As a result, patentees could credibly seek to exact royalties for nearly all improvements, whether literally infringing or not. Improvers, meanwhile, would need to think twice before refusing such demands. To a greater

---

subject matter, see Robert A. Kreiss, *Patent Protection for Computer Programs and Mathematical Algorithms: The Constitutional Limitations on Patentable Subject Matter*, 29 N.M. L. Rev. 31 (1999).

44. *See* Cohen, *supra* note 5, at 1169-70; Alan P. Klein, *Reinventing the Examination Process for Patent Applications Covering Software-Related Inventions*, 13 J. Marshall J. Computer & Info. L. 231 (1995); Merges, *supra* note 3, at 588-605; Stern, *supra* note 43, at 395.

45. An exception is Richard H. Stern, *On Defining the Concept of Infringement of Intellectual Property Rights in Algorithms and Other Abstract Computer-Related Ideas*, 23 AIPLA Q.J. 401 (1995). Even this early effort ends up focusing primarily on questions of patentable subject matter, however.

46. *E.g.*, Maureen A. O'Rourke, *Towards a Fair Use Defense in Patent Law*, 100 Colum. L. Rev. 1177 (2000); Robert P. Merges, *Who Owns the Charles River Bridge? Intellectual Property and Competition in the Software Industry* (2000) (on file with both authors).

47. Edmund W. Kitch, *The Nature and Function of the Patent System*, 20 J.L. & Econ. 265 (1977).

degree than ever before, second-comers would need permission to develop and market their innovations.

Kitch argued that a prospect system would produce both a more efficient allocation of resources to technical problems and greater overall progress. First, the system would prevent unnecessary decreases in social wealth by minimizing wasted or redundant efforts by competing improvers.[48] Since patents impose costs on society, a crucial assumption underlying this argument is that the opportunity costs generated by competing improvers exceed the deadweight losses that broad patents would generate.[49] Second, Kitch argued that a prospect system would maximize social wealth by ensuring both optimal incentives to commercialize the invention and the optimal allocation of licenses to develop improvements.[50] This argument is based on a set of assumptions about the rational behavior of prospect owners and improvers. It assumes that owners can readily identify, and would readily license, successful improvers; that the gains from coordination would outweigh the costs of any strategic behavior by owners and improvers; and that the initial allocation of stronger property rights to the prospect owner would not adversely affect improvers' incentives (or that an overall increase in productivity would outweigh any such adverse effect).

The prospect theory of optimal patent scope has both adherents and critics.[51] We take no position here on the abstract merits of the theory, or the question whether it might be sound as applied to some other class of inventions. We believe, however, that a shift of this magnitude in the operation of the patent law as applied to software should not go unremarked. Before adopting, or acceding to, a "prospect" system for software patents, it is important to ask whether such a system represents good policy for software innovation.

Whether a prospect approach is the right one for the software industry depends on whether Kitch's assumptions about relative costs and incentive effects are right, given the conditions in that industry. What are the patterns of innovation, and who are the innovators? How do technical constraints, such as interoperability requirements, and economic constraints, such as network effects, affect innovative patterns and practices? Is the class of

---

48. *Id.* at 276, 278.

49. A patent prevents some who would otherwise want to use the patented invention at a competitive price from doing so. This effect is termed "deadweight loss."

50. Kitch, *supra* note 47, at 276-78.

51. For adherents, see Mark F. Grady & Jay I. Alexander, *Patent Law and Rent Dissipation*, 78 Va. L. Rev. 305 (1992). For critics, see Seth A. Cohen, *To Innovate or Not to Innovate, That Is the Question: The Functions, Failures, and Foibles of the Reward Function Theory of Patent Law in Relation to Computer Software Platforms*, 5 Mich. Telecomm. & Tech. L. Rev. (1998), *at* http://www.law.umich.edu/mttlr/volfive/cohen.html; Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 Tex. L. Rev. 989 (1997); Robert P. Merges & Richard R. Nelson, *On the Complex Economics of Patent Scope*, 90 Colum. L. Rev. 839 (1990).

potential inventors and improvers small and homogeneous or is it large and heterogeneous? The remainder of this Article evaluates the effects of patent doctrine on software innovation in light of these and other considerations. In Parts II and III, we conclude that the particular characteristics of innovation within the software industry militate against such an approach, and that patents should be construed narrowly to avoid stifling progress.

II
Reverse Engineering Patented Software

Courts and scholars have devoted an enormous amount of time and effort to discussing the practice of reverse engineering computer software.[52] That discussion has primarily taken place under the aegis of trade secret and copyright laws because historically it was those laws that protected computer programs.[53] As we explain in Part II.A, although reverse

52. Reverse engineering of software, also called "decompilation," involves working backwards from object code to produce a simulacrum of the original source code. Andrew Johnson-Laird, *Software Reverse Engineering in the Real World*, 19 U. Dayton L. Rev. 843 (1994).

53. Virtually all recent court decisions have endorsed reverse engineering in some circumstances. *E.g.*, Sony Computer Entm't, Inc. v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000); DSC Communications Corp. v. DGI Techs., Inc., 81 F.3d 597, 601 (5th Cir. 1996); Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1539 n.18 (11th Cir. 1996); Lotus Dev. Corp. v. Borland Int'l, Inc. 49 F.3d 807, 817-18 (1st Cir. 1995) (Boudin, J., concurring); Atari Games Corp. v. Nintendo of America, Inc., 975 F.2d 832, 843-44 (Fed. Cir. 1992); Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510, 1527-28 (9th Cir. 1992); Vault Corp. v. Quaid Software Ltd., 847 F.2d 255, 270 (5th Cir. 1988); DSC Communications Corp. v. Pulse Communications, Inc., 976 F. Supp. 359 (E.D. Va. 1997); Mitel, Inc. v. Iqtel, Inc., 896 F. Supp. 1050 (D. Colo. 1995), *aff'd on other grounds*, 124 F.3d 1366 (10th Cir. 1997); *cf.* DSC Communications Corp. v. Pulse Communications, Inc., 170 F.3d 1354 (Fed. Cir. 1999) (acknowledging the right to reverse engineer for some purposes, but holding it unjustified in that case). On the other hand, a few early decisions rejected compatibility as a justification for copying. *E.g.*, Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983); Digital Communications Ass'n v. Softklone Distrib. Corp., 659 F. Supp. 449 (N.D. Ga. 1987).

As with courts, the overwhelming majority of commentators endorse a right to reverse engineer copyrighted software, at least for certain purposes. *See, e.g.*, Jonathan Band & Masanobu Katoh, Interfaces on Trial: Intellectual Property and Interoperability in the Global Software Industry 167-226 (1995); Cohen, *supra* note 5; Lawrence D. Graham & Richard O. Zerbe, Jr., *Economically Efficient Treatment of Computer Software: Reverse Engineering, Protection, and Disclosure*, 22 Rutgers Computer & Tech. L.J. 61 (1996); Dennis S. Karjala, *Copyright Protection of Computer Software, Reverse Engineering, and Professor Miller*, 19 U. Dayton L. Rev. 975, 1016-18 (1994); Maureen A. O'Rourke, *Drawing the Boundary Between Copyright and Contract: Copyright Preemption of Software License Terms*, 45 Duke L.J. 479, 534 (1995); David A. Rice, Sega *and Beyond: A Beacon for Fair Use Analysis . . . At Least as Far as It Goes*, 19 U. Dayton L. Rev. 1131, 1168 (1994); Pamela Samuelson, *Fair Use for Computer Programs and Other Copyrightable Works in Digital Form: The Implications of* Sony, Galoob *and* Sega, 1 J. Intell. Prop. L. 49 (1993); Tyler G. Newby, Note, *What's Fair Here Is Not Fair Everywhere: Does the American Fair Use Doctrine Violate International Copyright Law?*, 51 Stan. L. Rev. 1633, 1657-58 (1999); Timothy Teter, Note, *Merger and the Machines: An Analysis of the Pro-Compatibility Trend in Computer Software Copyright Cases*, 45 Stan. L. Rev. 1061 (1993) (arguing that the value of computer programs depends on interoperability); *see also* Pamela Samuelson & Suzanne Scotchmer, *The Law and Economics of Reverse Engineering* (working paper 2000) (on file with authors) (suggesting that reverse engineering should be legal when it promotes interoperability, but not when it permits free riding).

engineering of most types of patented inventions does not constitute infringement, the reverse engineering of patented software may be an exception. Part II.B discusses the reasons that the ability to reverse engineer software are important to the overall health of the software industry. In Part II.C, we explore the various ways in which courts or, as a last resort, Congress, might correct this software-specific anomaly in the patent law's reach.[54]

A. Software-Specific Barriers to Lawful Reverse Engineering of Patented Inventions

The intellectual property regimes that have traditionally protected software permit reverse engineering. Under trade secret law, there is no question that reverse engineering is legal.[55] Indeed, the Supreme Court has made it clear that the continued presence of a reverse engineering exception in trade secret law is necessary to avoid federal preemption.[56] Software vendors who rely on trade secret law, then, must accept the possibility that a consumer will reverse engineer their publicly-distributed object code and discover the secrets contained in the program.

While there is no express statutory provision in the copyright laws permitting reverse engineering, virtually every court to consider the issue has concluded that there is a right to reverse engineer a copyrighted program for at least some purposes. The source of that right is generally considered to be the fair use doctrine,[57] though reverse engineering finds some

---

For a contrary view, see generally Anthony L. Clapes, *Confessions of an Amicus Curiae: Technophobia, Law and Creativity in the Digital Arts*, 19 U. Dayton L. Rev. 903 (1994) (contending that there should be no right to reverse engineer software), and Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU?*, 106 Harv. L. Rev. 977 (1993) (same).

For a discussion of the history of copyright protection of software, see generally Pamela Samuelson, *CONTU Revisited: The Case Against Copyright Protection for Computer Programs in Machine-Readable Form*, 1984 Duke L.J. 663 (1984).

54. There are other potential threats to the reverse engineering right, notably contract law, and other areas in which reverse engineering is well established, notably the Semiconductor Chip Protection Act, 17 U.S.C. § 906(a), and the Digital Millennium Copyright Act, 17 U.S.C. § 1201(f). These issues are beyond the scope of this Article.

55. *E.g.*, Unif. Trade Secrets Act § 1, cmt., 14 U.L.A. 438-39 (1990); Chicago Lock Co. v. Fanberg, 676 F.2d 400 (9th Cir. 1982); Restatement of Torts § 757 cmt. f (1939); Restatement (Third) Unfair Competition § 43 (1995). The new federal criminal trade secrets statute, by contrast, is silent on the subject of reverse engineering. 18 U.S.C.A. §§ 1831-1839 (1984 & Supp. 2000); *see also* James H.A. Pooley et al., *Understanding the Economic Espionage Act of 1996*, 5 Tex. Intell. Prop. L.J. 177, 195-97 (1997) (arguing that the EEA might be construed to prohibit reverse engineering, but that it should not be).

56. *See* Bonito Boats, Inc. v. Thunder Craft Boats, Inc., 489 U.S. 141, 160 (1989); Kewanee Oil Co. v. Bicron Corp., 416 U.S. 470 (1974) (trade secret law not preempted by patent because the reverse engineering exception weakens trade secret law sufficiently that it does not interfere with patent policy).

57. 17 U.S.C. § 107 (1994). For cases finding a right to reverse engineer under fair use principles, see *supra* note 53.

support in the copyright misuse doctrine as well.[58] Courts have not determined that all reverse engineering is necessarily fair use; rather, as required by general principles of fair use, they have engaged in a case-by-case inquiry into the purposes and effects of the defendant's conduct. Reverse engineering has been held lawful primarily when used for laudable competitive purposes, such as producing new works that compete with the copyrighted original, producing products for downstream markets that are compatible with the copyrighted original, and obtaining access to uncopyrighted ideas, facts, or other material "locked up" within a copyrighted work.[59] Because reverse engineering is costly, this legal rule does not foreclose the possibility of a licensing arrangement. But it does prevent a potential licenser from refusing to deal at all, and it imposes a natural upper limit—the cost of reverse engineering—on what a licensee will be willing to pay.[60]

The introduction of patent protection for computer software threatens to change the equation, however. The patent statute includes no express provision allowing reverse engineering, nor is there any judicially-developed exception akin to copyright's fair use doctrine that might permit it. In theory, an express provision authorizing reverse engineering would be superfluous if the enabling disclosures required to secure a patent were sufficiently strong.[61] However, the Federal Circuit does not require would-be patentees of software inventions to disclose the implementing source code, or indeed very much at all about their inventions.[62] Accordingly, software

---

58. For a discussion of reverse engineering under principles of copyright misuse, see, for example, Cohen, *supra* note 5; O'Rourke, *supra* note 53, at 550; James A.D. White, *Misuse or Fair Use? That Is the Software Copyright Question*, 12 Berkeley Tech. L.J. 251, 287-88 (1997). For general background on the copyright misuse doctrine, see 2 Paul Goldstein, Copyright §§9:38-1 to 9:39 (2d ed. 1998); Marshall Leaffer, *Engineering Competitive Policy and Copyright Misuse*, 19 U. Dayton L. Rev. 1087 (1994); and Mark A. Lemley, *Beyond Preemption: The Law and Policy of Intellectual Property Licensing*, 87 Calif. L. Rev. 111, 151-58 (1999).

59. *E.g.*, Sony Computer Entm't, Inc. v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000) (holding that it was lawful to reverse engineer a video game system as an intermediate step to creating a computer program that would allow games designed for that system to run on a PC); Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1539-40 n.18 (11th Cir. 1996) (endorsing the use of reverse engineering to gain access to the unprotectable ideas in a program, as well as access to copyrighted expression that might be used fairly); Sega Enters. Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1992) (holding that reverse engineering was lawful where necessary to make an independently created video game work with the plaintiff's game system); Atari Games Corp. v. Nintendo of America, Inc., 975 F.2d 832, 843-44 (Fed. Cir. 1992) (same); Vault Corp. v. Quaid Software Ltd., 847 F.2d 255 (5th Cir. 1988) (holding that reverse engineering was lawful when done in order to make a product that defeated the plaintiff's copyrighted encryption product). *But see* DSC Communications Corp. v. Pulse Communications, Inc., 170 F.3d 1354 (Fed. Cir. 1999) (rejecting a reverse engineering claim on the particular facts before it).

60. *See* Samuelson & Scotchmer, *supra* note 53.

61. 35 U.S.C. § 112 (1994) (requiring patent applicants to describe their invention in such detail as to enable others to make and use it).

62. *Infra* notes 85-87 and accompanying text.

patents present unique obstacles to consummation of the patent law's traditional rights-for-disclosure bargain with the public.

The specific reverse engineering techniques commonly used for software may also raise some infringement problems that are unique to software. The definition of infringement in the patent statute is extremely broad, encompassing anyone who "makes, uses, sells, offers to sell, or imports" a patented product.[63] Reverse engineering a patented computer program by decompiling[64] it likely fits within this broad category of prohibited conduct, at least where the program itself is claimed as an apparatus. Reverse engineering clearly constitutes a "use" of the patented software, though owners of a particular copy of the program surely have the right to use it.[65] More significantly, decompilation may also constitute "making" the patented program by generating a temporary yet functional copy of it in RAM memory,[66] and, in certain instances, a longer-term (though still "intermediate") copy in more permanent memory.[67] Those copies probably constitute patent infringement, unless protected by some defense.[68]

---

63. 35 U.S.C. § 271(a) (1994).

64. We are concerned in this Part primarily with reverse engineering by "decompilation," that is, working backwards from the object code to construct a simulacrum of the source code. Other forms of reverse engineering, such as "black-box" reverse engineering, which infers details about a program's structure by testing its response to different inputs, do not involve making even temporary copies of the program, though they certainly involve "using" it. Our subsequent references to "reverse engineering" should be understood to refer to decompilation, not to black-box reverse engineering.

65. On the implied license and exhaustion doctrines that confer such a right, see *infra* Part II.C.2.

66. It seems clear that generating even temporary instantiations of a patented product "make" that product for purposes of patent infringement. This principle is firmly established in the pharmaceutical context, where courts have held that a patent is infringed when the patented product is generated by metabolization of a different drug within the human body, and that chemical "intermediates" temporarily generated in the course of making a final product infringe a patent covering those intermediates. *E.g.*, Hoechst-Roussel Pharm., Inc. v. Lehman, 109 F.3d 756, 759 (Fed. Cir. 1997); Zenith Labs v. Bristol Myers Squibb, 19 F.3d 1418, 1422 (Fed. Cir. 1994); *see also* Keith E. Witek, *Software Patent Infringement on the Internet and on Modern Computer System—Who Is Liable for Damages?*, 14 Santa Clara Computer & High Tech. L.J. 303 (1998) (arguing that since patent law lacks a fixation requirement, even near-instantaneous duplication of patented software is a prohibited "making" of the patented product).

Mahajan argues that reverse engineering for valid social purposes (compatability, competition or study) may be necessary, and likely does not constitute patent infringement. Anthony J. Mahajan, Note, *Intellectual Property, Contracts, and Reverse Engineering After ProCD: A Proposed Compromise for Computer Software*, 67 Fordham L. Rev. 3297, 3317-18 (1999). However, we think Mahajan has confused the result the law should reach with the result a court likely would reach by applying the statute.

67. Thus, an article of manufacture claim to a particular program "encoded on a computer hard drive" might be infringed by a reverse engineered copy temporarily stored on a computer hard drive.

68. One possible argument that the copies are noninfringing is that most copies made during the reverse engineering process are nonfunctional, either because they are only partial or because they are converted to assembly language or source code form. Theoretically, a source code readout of a computer program could be considered a description of the invention, rather than a copy of the invention itself. Nonetheless, decompilation also involves the generation of object code "copies" of the patented program, at least in RAM.

We note here that "software patents" are not a unitary phenomenon; thus, reverse engineering will not constitute infringement in all cases where it is employed. Parsing the question of whether reverse engineering "makes" a copy of the invention covered by a software patent requires, first, that we specify what we mean by a software patent. As Dennis Karjala has observed, software patents fall into two basic types: "pure" software patents claiming improvements in programming or inventions embodied wholly in a program, and "computer-related" inventions in which the claim is for a machine or process that happens to use a computer program.[69] To date, the latter have been more common, but we are seeing more and more pure software claims. An invention that includes software only as one component in a larger machine or process is unlikely to be "made" if the software component is reverse engineered. However, pure software inventions can be made in their entirety during the reverse engineering process. This is especially true of patents on inventions that are embedded in a larger program, such as a patent on a "system" of sorting data or dynamically linking items in a list.

Whether reverse engineering infringes a patent will further depend on the way the claim is written. Software inventions can be claimed as a process (a series of steps for accomplishing a result), an article of manufacture (the program itself, often embodied in a tangible item such as a floppy disk), or an apparatus (a machine, device, or system that performs a particular function).[70] Most clearly, reverse engineering a computer program will involve the making or using of a pure software invention covered by an article of manufacture or apparatus patent, because those patents cover the program itself rather than some use of the program. If the patentee has a process claim, whether reverse engineering will constitute infringement depends on what that claim covers. A process claim that is internal to the software, that is, one whose steps involve the internal operation of the program may be "used" automatically when the program is run or tested, or even when it is loaded into RAM. By contrast, an "external" process claim that requires the use of the program to perform some function in the real world probably would not be "used" during the process of reverse engineering. Because the patent with an external claim will be written to cover a process of generating some outcome or performing some function in the real world, only someone who actually performs the stated function will

---

69. Karjala, *supra* note 5, at 60-63.

70. 35 U.S.C. § 101 (1994) (permitting patents for a "process, machine, manufacture, or composition of matter"). Where part of a claim is written in "means-plus-function" format, determining the scope of the claim will require reference to the structure actually disclosed in the patent specification. Such claims are particularly common in software cases. *See* Mark D. Janis, *Who's Afraid of Functional Claims? Reforming the Patent Law's § 112, ¶ 6 Jurisprudence*, 15 Santa Clara Computer & High Tech. L.J. 231, 235 (1999). For an explanation of means-plus-function claims, see 35 U.S.C. § 112 ¶ 6 (1994).

infringe the method claim. The temporary copies generated by reverse engineers likely will not perform this function, and therefore will not infringe.

In short, some, but not all, software patent claims will raise the issue we discuss in this Part.[71] In the remainder of Part II, we are discussing only that subset of software patents for which the reverse engineering problem arises. For this class of patents, probably the majority of true software patents, there is reason to believe that applying patent law to software significantly changes the rules of the game for some would-be reverse engineers. This problem is far from hypothetical. In 1999, Sony sued Connectix for copyright infringement based on Connectix's successful effort to reverse engineer the Sony PlayStation and produce an emulator that would run Sony video games on a Macintosh computer. The Ninth Circuit held that Connectix's reverse engineering did not violate copyright law because it constituted fair use.[72] Less than a week later, Sony filed another lawsuit charging that Connectix's reverse engineering efforts constituted patent infringement, precisely the argument this section addresses. In 2000, Sony filed a similar patent lawsuit against Bleem, again after losing on its copyright infringement claims.[73] Meanwhile, Microsoft has asserted patents on a popular video file format to block distribution of an open source version developed by reverse engineering.[74]

B. Innovation and Reverse Engineering: An Industry-Based Analysis

The wisdom of permitting reverse engineering of software has been debated extensively in the last two decades.[75] We do not intend to rehash all those arguments here, though we think it clear that advocates of reverse engineering have the better part of the argument. Briefly, reverse engineering is an important means of preserving competition between different products and of preserving compatibility between products. In markets characterized by network effects, such as software, this latter objective is particularly important.[76]

---

71. Karjala argues that there is no reason to treat a software invention differently depending on the form in which it appears, because the invention lies in the methodology. Karjala, *supra* note 5, at 67-68. We are inclined to agree. However, for the reasons we suggest in this Part, we fear that the law will treat these different forms differently. Over time, moreover, this differential treatment may create incentives to draft claims in forms that will cover reverse engineering.

72. Sony Computer Entm't, Inc. v. Connectix Corp., 203 F.3d 596, 608 (9th Cir. 2000).

73. Sony Computer Entm't Am. v. Bleem, LLC, 214 F.3d 1022 (9th Cir. 2000); Bloomberg News, *Sony Sues Another Software Firm Over PlayStation Emulator* (May 18, 2000), *at* http://news.cnet.com/news/0-1006-202-1896822.html.

74. Andy Tai, *Microsoft Patents ASF Media File Format, Stops Reverse Engineering* (June 5, 2000), *at* http://www.advogato.org/article/101.html.

75. *Supra* note 53 (citing commentators).

76. Network effects exist where the value a user derives from a product is a positive function of how many others use the same product. Thus, telephony is a network market because a user's telephone becomes more valuable as more and more users buy telephones. On the implications of network effects,

The nexus among intellectual property, compatibility, and network effects is quite strong. To the extent that intellectual property rights confer ownership interests in a strong network standard, they may create durable market power in network markets. Conversely, the existence of compatibility between products or standards can in certain circumstances lower entry barriers created by network effects. The existing reverse engineering right afforded by the copyright and trade secret laws is particularly important in such markets because it facilitates competition within a network standard in cases in which competition between standards is either impossible or undesirable.[77]

These general arguments for permitting reverse engineering are strong, but we think the case for permitting reverse engineering of patented software is even stronger. Four additional policies specific to the patent law militate in favor of a limited reverse engineering right. Reverse engineering promotes the fundamental patent policies of disclosure and enablement, ensures that patents will not be leveraged to protect unprotectable components of software, preserves the balance sought by the intellectual property system as a whole, and also helps patentees enforce their rights.

---

see, for example, Michael L. Katz & Carl Shapiro, *Network Externalities, Competition, and Compatibility*, 75 Am. Econ. Rev. 424 (1985), and Mark A. Lemley & David McGowan, *Legal Implications of Network Economic Effects*, 86 Calif. L. Rev. 479 (1998). Software is characterized by network effects because widespread use facilitates interaction between different programs. *Id.* at 491-92. On the importance of compatibility in the presence of network effects, see Joseph Farrell & Michael L. Katz, *The Effects of Antitrust and Intellectual Property Law on Compatibility and Innovation*, 43 Antitrust Bull. 609 (1998).

77. Competition between potential standards may be undesirable in a strong network market because it can delay the adoption of a network standard. If the world were divided into two incompatible telephone networks of approximately equal size, for example, consumers would be worse off than with a monopoly phone system, because either network would only allow them to reach half of the population. If the network effects are strong enough, the harm from splitting or even delaying convergence upon a single standard will outweigh the value to competition between the potential standards on the intrinsic merits. *See generally, e.g.*, Mark A. Lemley, *Antitrust and the Internet Standardization Problem*, 28 Conn. L. Rev. 1041 (1996); Mark A. Lemley & David McGowan, *Could Java Change Everything? The Competitive Propriety of a Proprietary Standard*, 43 Antitrust Bull. 715 (1998).

Doug Lichtman has recently argued that facilitating competition in goods complementary to a network market is actually undesirable, because it results in a price that is too high given the network effects. He proposes that the network monopolist be permitted to control the market for complementary goods in order to coerce a lower price in that market. Douglas Lichtman, *Property Rights in Emerging Platform Technologies*, 29 J. Legal Studies 615 (2000). If Lichtman is correct—and we are not persuaded that any system manufacturer that has actually sought to control complementary goods has done so in order to reduce prices—his argument would be a reason to oppose reverse engineering in one specific class of cases: complementary goods to strong network markets. *But see* Jeffrey Church & Neil Gandal, *Systems Competition, Vertical Merger, and Foreclosure*, 9 J. Econ. & Mgmt. Strategy 1 (2000) (arguing that control by a hardware manufacturer over complementary software goods leads to monopolization of the complementary goods and higher prices).

1. Access to the Patented Invention

To an even greater extent than copyright law, patent law anticipates and even depends on one party improving another party's invention.[78] The patent statute itself expressly contemplates that "improvements" to other inventions are themselves a patentable category of invention,[79] and even invites patent claims that declare their "subservience" to a previously patented invention.[80] More importantly, patent law has developed doctrines that deal specifically with the circumstance in which one party's invention infringes another's patent, and yet at the same time represents an improvement on the first patented invention. These doctrines, denominated the "blocking patents" rule and the "reverse doctrine of equivalents," reward improvers even though their improvement infringes on a prior patent.[81] Indeed, the reverse doctrine of equivalents even excuses literal infringement if the infringer has radically improved the invention.[82]

These doctrines are premised on access by improvers to the underlying technology they can improve. Some inventions are readily apparent once embodied in a product; think of the paper clip, for example.[83] Improvers do not need to reverse engineer the paper clip and figure out how it works in order to improve it; they just need to look at it.

---

78. *See generally* Lemley, *supra* note 51, at 1000-29. This is true for a variety of reasons, but most importantly because the efficient creation of new works requires the new creator to have access to and use of old works. *E.g.*, Richard R. Nelson & Sidney G. Winter, An Evolutionary Theory of Economic Change 130 (1982); Merges & Nelson, *supra note* 51; Nathan Rosenberg, *Factors Affecting the Diffusion of Technology*, 10 Explorations Econ. Hist. 3 (1972); Suzanne Scotchmer, *Standing on the Shoulders of Giants: Cumulative Research and the Patent Law*, 5 J. Econ. Perspectives 29, 29-31 (1991).

79. 35 U.S.C. § 101 (1994).

80. These are so-called Jepson claims, which identify the invention they are improving in the preamble. *See, e.g.*, Pentec, Inc. v. Graphic Controls Corp., 776 F.2d 309 (Fed. Cir. 1985).

81. *E.g.*, Scripps Clinic & Research Found. v. Genentech, Inc., 927 F.2d 1565, 1581 (Fed. Cir. 1991) (suggesting that a literally infringing device may nonetheless escape liability under the reverse doctrine of equivalents because it is a radical improvement on the patented technology); United States Steel Corp. v. Phillips Petroleum Co., 865 F.2d 1247, 1253 n.11 (Fed. Cir. 1989) ("Dominating patents are not uncommon."); Atlas Powder Co. v. E.I. duPont de Nemours & Co., 750 F.2d 1569 (Fed. Cir. 1984) (involving an example of the blocking patents rule, which allows an infringing invention to receive its own improvement patent). For a more complete explication of the blocking patents and reverse doctrine of equivalents rules, see, for example, Lemley, *supra* note 51, at 1007-13; Robert Merges, *Intellectual Property Rights and Bargaining Breakdown: The Case of Blocking Patents*, 62 Tenn. L. Rev. 75 (1994) [hereinafter Merges, *Bargaining Breakdown*]; Robert P. Merges, *A Brief Note on Blocking Patents and Reverse Equivalents: Biotechnology as an Example*, 73 J. Pat. & Trademark Off. Soc'y 878 (1991) [hereinafter Merges, *A Brief Note*].

82. *Scripps Clinic & Research Found.*, 927 F.2d at 1581 (citing Graver Tank & Mfg. Co. v. Linde Air Prod. Co., 339 U.S. 605, 608-09 (1950)) (holding that a radically improved method of isolating drug using recombinant DNA might be excused from infringement).

83. *E.g.*, U.S. Patent No. 5,179,765 (issued Jan. 19, 1993) (granting patent to a "Plastic Paper Clip").

Patentable inventions in software, however, generally do not have these characteristics.[84]

Generally, patent law solves this access problem by requiring that patentees publish to the world a description of their invention sufficient to enable one of ordinary skill in the art to make and use it, as well as their "best mode" of implementing the invention.[85] Indeed, this disclosure "bargain" between patentees and the public is central to patent policy.[86] For software patents, however, a series of recent Federal Circuit decisions has all but eliminated the enablement and best mode requirements.[87] The result

---

84. Samuelson and her colleagues argue that certain features of computer programs are readily apparent to competitors, and therefore vulnerable to copying. Samuelson et al., *supra note* 5, at 2333. Their argument, however, is dependent not only on the vulnerability of programming innovations to casual inspection, but also on the ability of competitors to reverse engineer and analyze the design know-how lying "near the surface" of a program. *Id*. at 2335-37. If patent law precludes reverse engineering, it also precludes this sort of knowledge. It is true that certain types of computer program innovations, particularly user interfaces, are necessarily available to even the casual user, at least in part. But we doubt that these innovations are either the most significant parts of a new computer program or the most likely to be patented. Further, those innovations for which precise understanding is most important (such as application program interfaces) are also those which will not be available to casual inspection.

85. 35 U.S.C. § 112 ¶ 1 (1994).

86. One classic justification for having a patent system is to encourage inventors to disclose their ideas to the public, who will benefit from this new knowledge once the patent expires. Kewanee Oil Corp. v. Bicron Corp., 416 U.S. 470, 489 (1974) (referring to the "federal interest in disclosure" embodied in the patent laws); *see also* Edith Tilton Penrose, The Economics of the International Patent System 31-34 (1951).

87. In recent years, the Federal Circuit has held that software patentees need not disclose source or object code, flowcharts, or detailed descriptions of the patented program. Rather, high-level functional description is sufficient to satisfy both the enablement and best mode doctrines. *See* Fonar Corp. v. General Electric Co., 107 F.3d 1543, 1549 (Fed. Cir. 1997); *see also* Graham & Zerbe, *supra* note 53, at 96-97; Mahajan, *supra* note 66, at 3317. The Federal Circuit reasons that "the conversion of a complete thought . . . into a language a machine understands is necessarily a mere clerical function to a skilled programmer." Northern Telecom, Inc. v. Datapoint Corp., 908 F.2d 931, 941-42 (Fed. Cir. 1990) (quoting *In re* Sherwood, 613 F.2d 809, 817 (1980)). Indeed, the Federal Circuit has gone so far as to hold that patentees can satisfy the best mode requirement for inventions implemented in software even though they do not use the terms "computer" or "software" anywhere in the specification. Robotic Vision Sys., Inc. v. View Eng'g, Inc., 42 U.S.P.Q.2d 1619 (Fed. Cir. 1997); *In re* Dossel, 42 U.S.P.Q.2d 1881 (Fed. Cir. 1997). To be sure, in these latter cases it would probably be obvious to one skilled in the art that the particular feature in question should be implemented in software. Still, it is remarkable that the Federal Circuit is willing to find the enablement requirement satisfied by a patent specification that provides no guidance whatsoever on how the software should be written. 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. The Federal Circuit's peculiar direction in the software enablement cases has effectively nullified the disclosure obligation in software cases.

A recent development in Federal Circuit jurisprudence may suggest another source for a robust disclosure obligation, however. The court has recently reinvigorated the written description requirement in § 112, ¶ 1, not only in biotechnology cases, *e.g.*, Regents of the University of California v. Eli Lilly & Co., 119 F.3d 1559 (Fed. Cir. 1997), but also in cases about mechanical inventions. *E.g.*, Gentry Gallery, Inc. v. Berkline Corp., 134 F.3d 1473 (Fed. Cir. 1998). Under those cases, a patent claim is invalid if the specification does not expressly describe what the claim covers, even if the specification

is that software patentees generally do not disclose much, if any, detail about their programs, and therefore there is no easy way to figure out what a software patent owner has built except to reverse engineer the program.

There are other industries in which reverse engineering is necessary to determine the characteristics of an invention, but reverse engineering in those industries probably would not be patent infringement. If a competitor buys a patented chemical from the patent owner, analyzing that chemical in the laboratory does not trigger any of the exclusive rights listed in section 271.[88] Similarly, peeling apart the layers of a semiconductor chip in order to determine its layout, while extraordinarily difficult,[89] does not involve copying the chip itself. But because the most effective way to reverse engineer software is to "decompile" it, and decompilation makes a copy of the patented software, this form of analysis may well be held illegal under patent law. Thus, software patent owners will get a windfall if they can prevent reverse engineering: the right to preclude access to their invention and therefore to prevent others from improving it, despite the clear intent of the patent statute to the contrary.

2. Access to Unpatented Components

If access to the patented invention is a central part of patent policy, an even more important tenet of patent policy prevents patent owners from locking up access to unpatented ideas that are in the public domain. Indeed, the Supreme Court has stated that it would be unconstitutional to use patent law to withdraw works from the public domain,[90] and the antitrust and patent misuse rules have gone to great lengths to prevent patentees from expanding a patent beyond its bounds.[91] Still other patent doctrines, such as the doctrines of dedication to the public domain[92] and prosecution history

---

gave sufficient information to enable the claim. If this development proves durable, it could mean that most software patents will be invalid for failure to describe the invention in any detail.

88. The mere use of a lawfully purchased product is not illegal. *Infra* notes 116-118 and accompanying text (discussing the exhaustion doctrine).

89. Some sense of the difficulty can be gained by reading *Brooktree Corp. v. Advanced Micro Devices*, 977 F.2d 1555 (Fed. Cir. 1992), detailing unsuccessful efforts to reverse engineer a chip built in the mid-1980s, and then realizing that under Moore's law (the capacity of chips doubles every 18 months) modern chips are about one thousand times as complex as the chips at issue in that case.

90. Graham v. John Deere Co., 383 U.S. 1, 6 (1966) (concluding that it would be unconstitutional to grant "patents whose effects are to remove existent knowledge from the public domain, or to restrict free access to materials readily available").

91. The doctrine of patent misuse is primarily directed at preventing patentees from expanding their patent beyond the scope of the statutory grant. *See* B. Braun Med., Inc. v. Abbott Labs., 124 F.3d 1419, 1426-27 (Fed. Cir. 1997). Several antitrust doctrines, including the prohibition on tying arrangements, serve the same purpose in the patent context. *See* Int'l Salt Co. v. United States, 332 U.S. 392, 395-96 (1947).

92. Maxwell v. J. Baker, Inc., 86 F.3d 1098 (Fed. Cir. 1996) (holding that ideas disclosed in a patent specification, but not claimed in the patent, are dedicated to the public). The continued vitality of the *Maxwell* case is in doubt, however, after the Federal Circuit's decision in *YBM Magnex, Inc. v.*

estoppel,[93] are based on the premise that the patentee must not be allowed to expand its monopoly beyond the scope of what is claimed.

Software patents will create just such an expansion in the absence of a reverse engineering right. While some software patents, notably those that are really computer-related inventions, cover an entire computer program, the majority of true software patents (and virtually all of the truly nonobvious innovations in software) cover only a single part of a computer program. The invention may relate to a component of the larger program, or a particular algorithm or subroutine, or even a process for getting from one stage to another, but the invention is unlikely to be coextensive with an entire computer program.[94] In short, it is wrong to speak of a commercial program as being "patented" in the same sense that we might say it is "copyrighted." More properly, the software vendor has patents that cover certain inventions contained in the program. Many parts of the program, however, are unpatented.

For reasons discussed in the previous Part, the only way to get access to the unpatented components of the program often will be to reverse engineer the program, and therefore to "make" a copy of the entire program (including the patented components).[95] This is particularly true because in most cases it will be impossible even to tell ex ante which portions of a program are patented. If reverse engineering is illegal, then patenting even a small part of one computer program can give the patentee effective control over all the ideas contained in the program. Indeed, patentees have periodically taken advantage of this fact by patenting "lock-out" devices and using the patent to try to deny access to the unpatented components of special-purpose operating systems.[96] Given the patent policy in favor of free access to public domain works, this is of significant concern.

3. The Intellectual Property Balance

A variety of doctrines historically have served to channel certain sorts of innovation (technical) into the patent sphere and other sorts (artistic)

International Trade Commission, 145 F.3d 1317 (Fed. Cir. 1998), which purported to limit *Maxwell* to its facts.

93. Warner-Jenkinson Co. v. Hilton Davis Chem. Co., 520 U.S. 17, 30-34 (1997) (explaining that patentee is estopped from asserting infringement of broader claim if patentee narrowed that claim in response to an objection from the PTO).

94. For a discussion of patents on particular components of a computer program, see Mark A. Lemley & David W. O'Brien, *Encouraging Software Reuse*, 49 Stan. L. Rev. 255, 294-97 (1997). In this respect, software patents are like patents in the semiconductor industry, where the patented invention is normally only a tiny portion of a full product. By contrast, in industries such as pharmaceuticals, the scope of a patent is normally coextensive with a commercial product.

95. For reasons explained in the prior Part, this is generally not a problem in other industries.

96. *E.g.*, Atari Games Corp. v. Nintendo of America, Inc., 975 F.2d 832, 843-44 (Fed. Cir. 1992); Cohen, *supra* note 5, at 1152-53.

into the copyright sphere.[97] That division between art and science, never perfect, has all but disintegrated in the software realm.[98] Patents have expanded outside the realm of technology, and copyright has expanded to protect the functional aspects of utilitarian works.[99] As patent and copyright law overlap more and more, it becomes critical that they take account of each other.[100]

Copyright and trade secret law both have strongly articulated policies permitting reverse engineering where it is undertaken for a legitimate social purpose. For patent law to ban reverse engineering of software would undermine the goals of both copyright and trade secret law. It is little consolation to a reverse engineer who is held liable for patent infringement that he or she cannot also be sued for copyright infringement and misappropriation of trade secrets. Because patent, copyright, and trade secret rights can coexist simultaneously in the same piece of software, intellectual property policy for software must be made with the combination of rights in mind. If the courts conclude that patent law does not permit reverse engineering, they have effectively nullified the contrary rule in copyright and trade secret law.[101] This potential nullification is amply demonstrated by

---

97. Among those doctrines are the historic prohibition on patenting business methods and printed matter, neither of which fit within the "technological arts" to which patent law historically has extended. For a discussion of these doctrines and their recent disavowal, see Durham, *supra* note 23, and John R. Thomas, *The Patenting of the Liberal Professions*, 40 B.C.L. Rev. 1139 (1999). For its part, copyright has used the idea-expression dichotomy to channel certain types of creativity into the copyright realm, and others into the patent realm. *See, e.g.*, Baker v. Selden, 101 U.S. 99 (1879) (noting that some parts of a copyrighted work were eligible for protection only under patent law).

98. *See generally* J.H. Reichman, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 Colum. L. Rev. 2432 (1994) (discussing the traditional division, and deviations from this "bipolar" structure).

99. The floodgates for non-technological patents were opened by *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998), which allowed the patenting of pure business methods. A number of patents had already issued for such non-technological concepts as methods of holding a golf putter, however. *See, e.g.*, U.S. Patent No. 5,776,016 (claiming a "Golf Putting Method"). Copyright protection for software necessarily involves protection for the functional aspects of what are essentially utilitarian works, and copyright law has struggled in cases like *Lotus Development Corp. v. Borland International, Inc.*, 49 F.3d 807 (1st Cir. 1995), to reconcile this fact with the limiting doctrines of copyright law. As a result, copyright protection is not really centered on the real source of value in a computer program, which is its useful behavior. *See, e.g.*, Samuelson et al., *supra* note 5, at 2350. A number of authors have suggested that neither patent nor copyright fits software particularly well. *E.g.*, Peter S. Menell, *Tailoring Legal Protection for Computer Software*, 39 Stan. L. Rev. 1329 (1987); Samuelson et al., *supra* note 5, at 2308.

100. Karjala, *supra* note 5, at 43-44.

101. For a detailed discussion of the overlap between copyright and patent in this area, see Karjala, *supra* note 5, and Dennis S. Karjala, *A Coherent Theory for the Copyright Protection of Computer Software and Recent Judicial Interpretations*, 66 U. Cin. L. Rev. 53 (1997); *cf.* Merges, *supra* note 46, at 16-17 (noting this problem, but suggesting that disaggregated ownership of software patents may result in collective rights organizations that promote interoperability, and therefore preclude the need to reverse engineer patented programs). Merges bases his argument on the fact that software patents protect only particular components of a program, rather than the program as a whole, and that patentees will therefore need to trade with each other to obtain rights. While this is undoubtedly true in certain industries, such as semiconductors, it remains to be seen whether a similar

Sony v. Connectix, discussed above.[102] There, Sony has engaged in an end-run around the Ninth Circuit decision on copyright infringement by filing a software patent infringement suit against the same act of reverse engineering that the court held legal under copyright law.

4. Litigation-Related Uses

Finally, a ban on reverse engineering interferes with legitimate litigation-related investigations. Ironically, such a ban may make it difficult or impossible to detect patent infringement. Many software inventions are internal to the program, and their use cannot be detected without parsing the code. A patent owner who suspects a rival of infringing such a software patent may have no choice but to reverse engineer the rival's software in order to gain the evidence it needs to file suit.[103] If that rival has its own patents on a separate aspect of the program, however, reverse engineering as part of a pre-filing investigation will itself infringe the rival's patents. At the least, this puts a new argument in the hands of a patent defendant; at most, it may deter meritorious patent infringement suits from ever being filed.[104]

A ban on reverse engineering will also limit investigations by potential infringers into the validity of the patent. Reverse engineering a program may be the only way to determine that a patentee failed to disclose its best mode. Alternatively, reverse engineering may disclose that a patented invention was in fact in use before a critical statutory bar date, and that the patent is therefore invalid.[105] Because source code is not published with the patent, reverse engineering may be the only way to investigate the workings of a patented program.

---

market will develop for software patents. *Cf.* Lemley & O'Brien, *supra* note 94 (noting the component-based nature of software patents, but suggesting that the law should promote the recombination of components by denying strong protection to the interfaces between them). Indeed, how the market develops may depend fundamentally on how broadly software patents are construed, a subject we take up in the next Part.

102. *See supra* note 72 and accompanying text.

103. While some evidence may be available in discovery once a suit is filed, the Federal Circuit has made it quite clear that a patentee cannot file suit based on a mere suspicion of infringement, but must have made a "reasonable inquiry" before filing suit. Indeed, to file suit without such an inquiry violates Rule 11. *E.g.*, View Eng'g, Inc. v. Robotic Visions Sys., 54 U.S.P.Q.2d 1179 (Fed. Cir. 2000); Judin v. United States, 110 F.3d 780 (Fed. Cir. 1997); Refac Int'l Ltd. v. Hitachi Ltd., 19 U.S.P.Q.2d 1855, 1858-59 (C.D. Cal. 1991) (imposing sanctions against a plaintiff who did not examine every accused device before filing suit). *But see* Vista Mfg., Inc. v. Trac-4, Inc., 15 U.S.P.Q.2d 1345, 1347-48 (N.D. Ind. 1990) (no "general rule that Rule 11 requires an infringement plaintiff to examine the defendant's product in all instances").

104. *Cf.* Pamela Samuelson, *Intellectual Property and the Digital Economy: Why the Anti-Circumvention Regulations Need to Be Revised*, 14 Berkeley Tech. L.J. 519, 543 (1999) (noting a similar problem with the new Digital Millennium Copyright Act, which makes it illegal to circumvent copy protections even to determine whether the protected work is infringing).

105. 35 U.S.C. § 102(b).

C. Creating a Right to Reverse Engineer Patented Software

Given the strong policy reasons to permit reverse engineering of patented software, it is worth taking a closer look at several defenses to patent infringement that might protect such activity. No court has yet considered whether reverse engineering of patented software is infringing; thus, no court has considered whether reverse engineering is protected by the experimental use, exhaustion, implied license, or misuse doctrines. It is not clear how such a defense would be resolved. The experimental use doctrine in particular appears to have been interpreted very narrowly, and the implied license and exhaustion rules may be too easy to undermine by contract. At the same time, we think the policy arguments in favor of permitting reverse engineering of patented software discussed in the previous Part are overwhelming. Reverse engineering enables competitors to develop noninfringing products, to develop new products that are compatible with existing standards, and to have access to the unprotected parts of patented programs. A right to reverse engineer patented software is consistent with the right to use other patented inventions once lawfully purchased, and with the way the copyright and trade secret laws treat software. Accordingly, we recommend that Congress legislate a reverse engineering defense if the courts do not recognize one.

1. Experimental Use

The patent statute itself contains only a narrow experimental use defense, and it is limited to circumstances clearly not relevant here.[106] However, there is also a non-statutory exception for experimental uses. Ever since Justice Story's decision in *Whittemore v. Cutter*,[107] it has been settled law that purely experimental uses were noninfringing. The court reasoned that "it could never have been the intention of the legislature to punish a man, who constructed such a machine merely for philosophical experiments, or for the purpose of ascertaining the sufficiency of the machine to produce its described effects."[108] Justice Story distinguished inventions made "with a design to use [them] for profit," however. The latter could not be thought "experimental" in nature.[109]

---

106. 35 U.S.C. § 271(e)(1) (1994) (allowing only experimental activity preparatory to the filing of a new product application before the Food and Drug Administration). The experimental use defenses to infringement should be distinguished from the doctrine of experimental use in 35 U.S.C. § 102(b) (1994), which excuses a delay in patenting by someone who is still experimenting with his or her invention. On the latter doctrine, see *Elizabeth v. Pavement Co.*, 97 U.S. 126 (1877).

107. 29 F. Cas. 1120 (C.C.D. Mass. 1813).

108. *Id*. at 1121.

109. *Id*.

This experimental-commercial distinction has been applied with increasing rigor over the years.[110] The result is to make the experimental use defense "truly narrow," and therefore of little use to most litigants.[111] In *Roche Products, Inc. v. Bolar Pharmaceutical Co.*, the Federal Circuit concluded that a use was not "experimental" within the meaning of the exception if it "has definite, cognizable, and not insubstantial commercial purposes."[112] Only if an experiment has no ultimate commercial purpose at all will it be protected under this doctrine.[113]

This is not the only possible conclusion, or even a wise one. Rebecca Eisenberg, in particular, has articulated a compelling argument for a broader vision of experimental use, one that encompasses scientific exploration leading to commercial but noninfringing end products.[114] If applied in software, such a vision would surely protect reverse engineers. We think a court probably should read the experimental use defense this way. But as currently interpreted by the courts, the experimental use defense will not aid reverse engineers who hope to make and sell noninfringing products.[115]

2. First Sale, Implied License, and Exhaustion

It is a well-established principle of patent law that a patentee's right to control the use of his patented goods does not extend beyond the first sale of a patented product. That is, a consumer who buys a patented product (or a product that necessarily uses a patented process) from the patentee has

---

110. For an excellent history of the research exemptions, see David L. Parker, *Patent Infringement Exemptions for Life Science Research*, 16 Hous. J. Int'l L. 615, 626-36 (1994).

111. *See* Roche Prods., Inc. v. Bolar Pharm. Co., 733 F.2d 858, 863 (Fed. Cir. 1984). For a discussion of the scope of the experimental use doctrine, see Lauren C. Bruzzone, *The Research Exemption: A Proposal*, 21 AIPLA Q.J. 52 (1993).

112. *Roche Prods.*, 733 F.2d at 863; *see also* Embrex v. Service Eng. Corp., 55 U.S.P.Q.2d 1161 (Fed. Cir. 2000) (holding that commerical enterprise's use of patented process in laboratory test did not qualify as experimental use); Pitcairn v. United States, 547 F.2d 1106, 1125-26 (Ct. Cl. 1976) (holding that experiments "in keeping with the legitimate business" of the accused infringer are not exempt from the patent laws).

113. *Roche Prods.*, 733 F.2d at 862 (quoting W. Robinson, The Law of Patents for Useful Inventions § 898 (1890)) ("where it is made or used as an experiment, whether for the gratification of scientific tastes, or for curiosity, or for amusement, the interests of the patentee are not antagonized"). One court offered a slightly broader reading of the doctrine of experimental use in *Giese v. Pierce Chemical Co.*, 29 F. Supp. 2d 33 (D. Mass. 1998). That court noted that use of a patented process for academic medical research might well be a protected experimental use, though it refused to decide the question on summary judgment. Even if this new interpretation is widely accepted, however, it will do little to help most reverse engineers in the software industry.

114. Rebecca S. Eisenberg, *Patents and the Progress of Science: Exclusive Rights and Experimental Use*, 56 U. Chi. L. Rev. 1017, 1078 (1989); *see* John H. Barton, *Reforming the Patent System*, 287 Sci. 1933 (2000). *But see* Jordan P. Karp, *Experimental Use as Patent Infringement: The Impropriety of a Broad Exception*, 100 Yale L.J. 2169 (1991).

115. Indeed, the judicial trend seems to be in the opposite direction. Judge Rader, concurring in *Embrex*, would have abolished the doctrine outright.

the right to use and resell that product without the patentee's approval.[116] Courts have developed two parallel doctrines that support such a right: the principles of exhaustion and implied license. These doctrines have similar consequences, but they stem from very different sources.[117]

The exhaustion doctrine finds its basis in the foundations of patent policy, which seeks not only to grant exclusive rights to patentees but also to limit those rights. Exhaustion represents one such limit on a patentee's right to control her invention: that control ceases with respect to a particular product once she has sold that product. In the words of the Supreme Court, "when the machine passes to the hands of the purchaser, it is no longer within the limits of the monopoly. It passes outside of it, and is no longer under the protection of the [patent laws]."[118] It is not the patent right itself that is exhausted, of course. The patentee retains the rights to prevent anyone else, including the buyer, from making, using, or selling additional copies of the patented item. But once the patentee has sold a particular product, its control over that particular product ends, and the general legal antipathy toward restraints on alienation takes over.

The doctrine of implied license impels courts to much the same conclusion: buying a product carries with it an implied right to use and resell the product.[119] Indeed, courts have frequently conflated the two doctrines.[120] But while patent exhaustion stems from inherent limits on the grant of the patent right, implied license is a doctrine of quasi-contract, and depends on the beliefs and expectations of the parties to the sales

---

116. United States v. Univis Lens Co., 316 U.S. 241, 249 (1942) ("An incident to the purchase of any article, whether patented or unpatented, is the right to use and sell it . . . ."); Glass Equip. Dev. v. Besten, Inc., 50 U.S.P.Q.2d 1300 (Fed. Cir. 1999) ("The first sale doctrine stands for the proposition that, absent unusual circumstances, courts infer that a patent owner has given up the right to exclude concerning a patented article that the owner sells."); Intel Corp. v. ULSI Sys. Tech., 27 U.S.P.Q.2d 1136 (Fed. Cir. 1993); Becton, Dickinson & Co. v. Eisele & Co., 86 F.2d 267, 270 (6th Cir. 1936) ("Once having sold patented articles, neither the patentee nor its licensee may exercise future control over them. They pass beyond the scope of the patentee's monopoly."); 5 Chisum, *supra* note 13, at § 16.03(2)(a). Similarly, when a patentee sells a product to be used in a patented process, the sale of the product normally carries with it an implied license to use the patented process. *Glass Equip. Dev.*, 50 U.S.P.Q.2d at 1302-03.

117. For an excellent discussion of both doctrines in historical context, see Mark D. Janis, *A Tale of the Apocryphal Axe: Repair, Reconstruction, and the Implied License in Intellectual Property Law*, 58 Md. L. Rev. 423 (1999), and Chisum, *supra* note 13, at § 16.03.

118. Bloomer v. McQuewan, 55 U.S. (14 How.) 539, 549 (1852); *see also* Goodyear v. Beverly Rubber Co., 10 F. Cas. 638, 641 (C.C.D. Mass. 1859) (stating that legal control of patented property passes to buyer after a valid sale).

119. General Elec. Co. v. United States, 572 F.2d 745, 784-85 (Ct. Cl. 1978) ("[I]t can be properly assumed that as part of the bargain the seller of a device incorporating a patented combination . . . authorizes the buyer to continue to use the device . . . .").

120. Janis, *supra* note 117, at 495 (noting instances of such confusion). *But cf.* Wang Labs., Inc. v. Mitsubishi Elecs. Am., Inc., 103 F.3d 1571 (Fed. Cir. 1997) (cataloguing various sorts of license and estoppel claims).

transaction.[121] It is most commonly applied in cases where the product sold by the patentee is not itself patented, but is necessary for use in a patented process.[122]

Both doctrines have traditionally drawn a distinction between using and reselling a particular copy of a patented product, which is permissible, and making a new copy of a patented product, which is not. Software patents undermine this distinction. It is impossible to use software without "making" a copy, at least temporarily, in the memory of a computer.[123] If the exhaustion and implied license doctrines do not protect the making of such temporary copies, those doctrines will effectively be nullified in the software context. No use of a purchased program would be permissible without express permission from the patentee.

We think that a reasonable court should reject this interpretation. Rather than focusing blindly on the distinction between making and using, courts dealing with software patents should look to the underlying policies behind the exhaustion and implied license principles. Permitting reasonable uses of the purchased software serves those underlying policies. Reverse engineering is such a use. It is legal under all other intellectual property laws, and has long been a favored tool of computer programmers. Further, reverse engineering of patented non-software products would unquestionably be lawful, for the simple reason that it would constitute only a "use" and not an impermissible "making" of the patented product.[124]

The next question is whether patentees can withdraw the protection of the exhaustion and implied license doctrines by refusing to permit buyers to reverse engineer their software. Here, exhaustion and implied license give potentially different answers. An implied license is, after all, a contractual vehicle; a license that is merely implied from the transaction's circumstances ordinarily can be disclaimed by an express statement to the

---

121. Janis, *supra* note 117, at 502-505 (noting the critical role intent of the parties plays in determining the scope of an implied license).

122. The exhaustion doctrine would not apply in such a circumstance, because there has been no "first sale" of a patented product.

123. This has been the subject of considerable litigation in the copyright arena. Most courts now hold that a temporary copy loaded in the RAM memory of a computer is "fixed" and therefore constitutes a new copy for copyright purposes. *E.g.*, MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 518 (9th Cir. 1993). While this is almost certainly the wrong conclusion, *see* Mark A. Lemley, *Dealing with Overlapping Copyrights on the Internet*, 22 U. Dayton L. Rev. 547, 551-52 & n.25 (1997) (cataloguing the critiques of *MAI*), an analogous conclusion seems self-evident in patent law. Because patent law has no fixation requirement at all, any reproduction of a patented program, no matter how temporary, arguably constitutes a "making" within the meaning of the statute. Witek, *supra* note 66, at 369-72.

124. Thus, we agree with Janis insofar as he objects to "device-oriented" results: there is no reason for the particular nature of software to change the effective legal rights buyers possess. Janis, *supra* note 117, at 492.

Of course, the exhaustion and implied license doctrines would only protect intermediate copying done as part of reverse engineering. The reverse engineer would still be obligated to ensure that its final product did not infringe the patent.

contrary.[125] Because the exhaustion doctrine is based in patent policy, however, and not the patentee's intent, it is harder to avoid by contract. Thus, in *Hewlett-Packard Co. v. Repeat-O-Type Stencil Manufacturing Corp.*,[126] H-P's patented printer came with various unilateral statements indicating H-P's intent that the printer's ink cartridge be discarded when empty. The court refused to conclude that the defendant engaged in patent infringement by refilling empty ink cartridges. It clearly grounded the limits on the patent right not in an implied license, but in a theory of exhaustion:

> The question is not whether the patentee at the time of sale intended to limit a purchaser's right to modify the product. Rather the purchaser's freedom to repair or modify its own property is overridden under the patent laws only by the patentee's right to exclude the purchaser from making a new patented entity. Each case turns on its own particular facts, but a seller's intent, unless embodied in an enforceable contract, does not create a limitation on the right of a purchaser to use, sell, or modify a patented product . . . . A noncontractual intention is simply the seller's hope or wish, rather than an enforceable restriction.[127]

Although *Hewlett-Packard* offers a strong endorsement of the exhaustion principle, other recent trends threaten to render the principle a nullity by expanding the contractual exception to swallow the rule. In particular, the court in *Mallinckrodt, Inc. v. Medipart, Inc.*[128] raised the possibility that the exhaustion doctrine could be avoided by the simple expedient of affixing a label to the patented product that read "single use only." The court reasoned that the exhaustion doctrine was subject to alteration by contract, and that the "label license" might be just such a contract.[129] One can reach that result only by mutilating contract law,[130] and

---

125. *E.g.*, Withington-Cooley Mfg. Co. v. Kinney, 68 F. 500, 506 (6th Cir. 1895) ("The duration and scope of a license must depend upon the nature of the invention and the circumstances out of which an implied license is presumed, and both must at last depend upon the intention of the parties."). *But cf.* Carborundum Co. v. Molten Metal Equip. Innovations, Inc., 72 F.3d 872, 877 (Fed. Cir. 1995) ("Whether there existed an implied license is a question of law.").

126. 123 F.3d 1445 (Fed. Cir. 1997).

127. *Id*. at 1453; *see also Carborundum*, 72 F.3d at 878 ("One party's unilateral expectations as to the scope of the implied license are irrelevant."). Janis criticizes *Hewlett-Packard* for ignoring the seller's intent, Janis, *supra* note 117, at 502-03, but we think it is precisely the right result. At least absent an enforceable contract to the contrary, the exhaustion doctrine gives buyers a legal right that sellers should not be able to defeat unilaterally.

128. 976 F.2d 700 (Fed. Cir. 1992).

129. *Id*. at 703-09; *accord* B. Braun Med., Inc. v. Abbott Labs., 124 F.3d 1419 (Fed. Cir. 1997).

130. The court suggested in a footnote that a label affixed to a product was a "form" triggering the battle of the forms, and that it therefore became part of the contract unless the other party objected within a reasonable time. It did not rule on the question, however. Even assuming U.C.C. Section 2-207 is the correct statute to apply to such a "form," the court simply misread the statute. First, the court's conclusion would require treating the patentee as the accepting rather than the offering party for Section 2-207 purposes, even though the opposite conclusion seems more logical. Second, it ignores Section 2-207(2)(b), which provides that such a term becomes part of the contract only if it does not

perhaps the suggestion that unilateral labeling trumps the exhaustion right will be ignored.[131] But if followed widely, such a policy of automatic licensing could signal the death of the exhaustion doctrine, at least in any case where the patentee is smart enough to unilaterally (or after the fact) characterize the sale as a limited license instead.[132] This is a particular danger in the software context, since software vendors have taken the position that all software is licensed rather than sold.[133] If this view were widely accepted, nothing would remain of the exhaustion doctrine in the software industry, since it would be so easy to condition the transaction on a shrinkwrap or clickwrap license to which the licensee might not even have prior access.[134]

A recent Federal Circuit decision involving the reverse engineering of software sheds some light on this question. In *DSC Communications Corp. v. Pulse Communication, Inc.*[135] the court held that negotiated agreements between DSC and a third party treated the transfer of software as a restricted license rather than a sale.[136] At the same time, the court noted that other copies of the software were purchased without restriction on the open market, and held that these copies were sold. Thus, the court applied different legal rules to the same software depending on the contractual

---

"materially alter" the deal. It is hard to argue that the license term at issue here was not a material alteration, since it vitiated the buyer's rights of reuse and resale completely. It is also worth questioning why Medipart, a company that was not in privity in any sense with Mallinckrodt, would be bound by the contract and therefore liable for patent infringement. Hon. Arthur J. Gajarsa et al., *How Much Fuel to Add to the Fire of Genius? Some Questions About the Repair/Reconstruction Distinction in Patent Law*, 48 Am. U. L. Rev. 1205, 1229-31 (1999).

131. In contrast to *Mallinckrodt*, see *Kendall Co. v. Progressive Medical Technologies, Inc.*, 85 F.3d 1570 (Fed. Cir. 1996). In *Kendall*, the court refused to enforce a similar "license" contained in the literature accompanying the product, concluding that the patentee's unilateral statement that only its replacement components could be used "did not have contractual significance." *Id.* at 1576; *cf.* Glass Equip. Dev. v. Besten, Inc., 174 F.3d 1337, 1342 (Fed. Cir. 1999) (noting that exhaustion is the normal state of affairs, and is avoided only in "unusual circumstances"). For trenchant criticism of *Mallinckrodt*, see James B. Kobak, Jr., *Contracting Around Exhaustion: Some Thoughts About the CAFC's* Mallinckrodt *Decision*, 75 J. Pat. & Trademark Off. Soc'y 550 (1993). Among other defects, Kobak notes that the decision conflates implied license, exhaustion, and antitrust cases into a single "confusing *melange*." *Id*. at 554.

132. *Cf.* Michael J. Swope, *Recent Developments in Patent Law: Implied License - An Emerging Threat to Contributory Infringement Protection*, 68 Temp. L. Rev. 281, 305-06 (1995) (suggesting that patentees should impose such restrictions).

It is possible that the doctrine of patent misuse would bar such contracts even if the exhaustion doctrine did not. *Cf.* Gajarsa et al., *supra* note 130, at 1226-29 (considering whether patent misuse would do this, and arguing that *Mallinckrodt* left the question open).

133. Mark A. Lemley, *Beyond Preemption: The Law and Policy of Intellectual Property Licensing*, 87 Calif. L. Rev. 111 (1999) (noting the efforts of software vendors to entrench this view by passing a new law, the proposed "Uniform Computer Information Transactions Act").

134. *See, e.g.*, Hill v. Gateway 2000, Inc., 105 F.3d 1147 (7th Cir. 1997) (a piece of paper unilaterally included in a box shipped to a consumer constituted an enforceable contract).

135. 170 F.3d 1354 (Fed. Cir. 1999).

136. *Id.*; Lemley, *supra* note 133.

circumstances surrounding its transfer.[137] While DSC involved copyright rather than patent law, it is not implausible that the court would do the same in a software patent case.[138]

The most plausible reading of the law in this area, then, is that it is sometimes possible to restrict resale or use of a patented product after first sale, but that such circumstances are "unusual" and must be clearly articulated in the contract.[139] At a minimum, we think the same principle should be extended to reverse engineering, traditionally considered a "use" of a purchased product, even if that reverse engineering requires the making of temporary copies of a computer program. Even if that is done, though, courts will have to confront the question of whether the exhaustion doctrine can in fact be overridden by contract.[140] One vehicle for confronting that question is the patent misuse doctrine.

3. Patent Misuse

The patent misuse doctrine might be enlisted to protect reverse engineering for compatibility purposes, just as the copyright misuse doctrine has been.[141] Patent misuse is an equitable defense to patent infringement that precludes patentees from "'impermissibly broaden[ing] the 'physical or temporal scope' of the patent grant with anticompetitive effect.'"[142] If a patentee has engaged in misuse, she will be prevented from enforcing her patent at all, at least until the misuse has been purged.[143] Patent misuse is frequently, but not entirely, coextensive with conduct that violates the antitrust laws.

Such a patent misuse argument would arise if a patent defense (such as exhaustion) were interpreted to protect reverse engineering, but the patentee conditioned the sale or license of the patented product on an agreement not to reverse engineer the product. In this situation, the question for misuse purposes would be whether such a condition served to "impermissibly broaden[] the 'physical or temporal scope' of the patent

---

137. *DSC Communication Corp.*, 170 F.3d at 1360-63.

138. This approach implicates the long-standing dispute over whether transactions in software are to be characterized as sales or licenses. For more detail on this dispute, see Lemley, *supra* note 133.

139. Glass Equip. Dev. v. Besten, Inc., 174 F.3d 1337, 1342 n.1 (Fed. Cir. 1999). This is consistent with the rule in the U.K. *See, e.g.*, Roussel-Uclaf v. Hockley, [1996] R.P.C. 441; Solar Thomson v. Barton, [1977] R.P.C. 537.

140. Gajarsa et al., *supra* note 130, at 1225-26 (noting this looming issue).

141. Alcatel USA, Inc. v. DGI Tech., Inc., 166 F.3d 772 (5th Cir. 1999); DSC Communications Corp. v. DGI Tech., Inc., 81 F.3d 597, 601 (5th Cir. 1996) ("DGI may well prevail on the defense of copyright misuse, because DSC seems to be attempting to use its copyright to obtain a patent-like monopoly over unpatented microprocessor cards.").

142. B. Braun Med., Inc. v. Abbot Labs., 124 F.3d 1419, 1426 (Fed. Cir. 1997) (quoting Windsurfing Int'l, Inc. v. AMF, Inc., 782 F.2d 995, 1001-02 (Fed. Cir. 1986)).

143. For general background on patent misuse, see 6 Chisum, *supra* note 13, § 19.04.

grant with anticompetitive effect."[144] We think that is precisely the effect of such an agreement. As noted above, preventing buyers from reverse engineering software not only prevents the noninfringing use of the patented component, but precludes any access to or use of unpatented portions of the same program.[145] A contract that effectively extends patent protection from one part of the program to cover unpatented parts should be vulnerable under principles of misuse.[146] Thus, if patent law does permit reverse engineering, as we think it should, a patentee should not necessarily be able to change that rule by contract. Doing so would run afoul of the purpose of the misuse doctrine, particularly where (as in the software industry) such "contractual" changes are likely to be ubiquitous, unbargained, and imposed unilaterally by vendors without any real notice.

4. New Legislation

In sum, there is some likelihood that existing patent defenses will be applied to excuse reverse engineering of patented software. As currently defined, the experimental use defense is too slim a reed to support a reverse engineering right. We believe, though, that a court faced with the question could easily decide that the patent exhaustion doctrine permits the buyer of patented software to use that software in a way that makes temporary copies of the program. Indeed, that seems the only logical conclusion consistent with the history and purpose of the exhaustion doctrine. From that conclusion, it would be only a small step to preclude contracts banning the reverse engineering of software, just as courts have done in the copyright and trade secret contexts. Yet the signals from the Federal Circuit regarding the continued viability of the exhaustion doctrine are mixed.[147]

---

144. *B. Braun Med.*, 124 F.3d at 1426 (quoting *Windsurfing Int'l*, 782 F.2d at 1001-02). A defendant also could argue that such a contract term was preempted because it effectively nullified a right granted the buyer under federal law. *Cf.* Vault Corp. v. Quaid Software Ltd., 847 F.2d 255, 270 (5th Cir. 1988) (preempting a no-reverse-engineering clause under copyright law). For a discussion of such federal policy preemption, see Lemley, *supra* note 133.

145. *See supra* notes 96-102 and accompanying text.

146. The analogy to the copyright misuse cases, especially *DSC Communications Corp.* and *Alcatel USA, Inc.*, is quite strong. In these cases, the copyright owner claimed that using a copyrighted circuit to test an interoperable but noninfringing circuit was copyright infringement, because the test necessarily made temporary copies of the copyrighted work. *DSC Communications Corp.*, 81 F.3d at 600; *Alcatel USA, Inc.*, 166 F.3d at 791. That is precisely the argument a plaintiff would make against reverse engineering in the patent context.

One difference between the copyright and patent contexts is the existence of 35 U.S.C. § 271(d). This section is reasonably construed as granting the patentee implicit power to control "non-staple" goods (that is, goods that have no use or value except in connection with the patent). *See* Dawson Chem. Co. v. Rohm & Haas Co., 448 U.S. 176 (1980). In some cases, one might argue that the unpatented part of a computer program was a non-staple good over which the patent should grant effective control, particularly if the valuable parts of the program were all patented. In other cases, however, the unpatented elements will turn out to be valuable in themselves, and therefore to be staple items of commerce.

147. *See infra* notes 126-140 and accompanying text.

Alternatively, and as a last resort, Congress could expressly legislate in this area. Some commentators have suggested that patent law should include a general fair use right;[148] if it did, that right would certainly encompass the limited protections for reverse engineering we suggest here. More likely, Congress could create a specific statutory right to reverse engineer patented software. Congress has already recognized the importance of reverse engineering by protecting reverse engineers under both the Semiconductor Chip Protection Act[149] and the Digital Millennium Copyright Act.[150] While we think that existing law, properly interpreted, can and should protect reverse engineering for legitimate purposes, Congress could guarantee such a right if the courts fail to do so.[151]

III
Designing Around Existing Software Patents

The freedom to reverse engineer, although vitally important to software developers, is only half the battle. For commercial software firms, the knowledge gained through reverse engineering is pointless unless it can be used to develop a marketable product. To identify the product development activities that are permissible and those that would require a license, the firm must consider the patented invention. It must attempt to determine the breadth of the claims and the scope that a court would give them in infringement litigation. This information will shape the firm's decisions about which innovative pathways to pursue, or whether to attempt to innovate around the patented invention at all. Here, we argue that the same characteristics of the software industry that require latitude for research use of patented software also require a narrow approach to questions of patent scope.

Once a software patent has been issued, the literal scope of its claims will be construed by the court as a matter of law in any infringement suit.[152] In theory, this process of claim construction determines the scope of the patent. In practice, however, the doctrine of equivalents is the primary tool available to courts and litigants for fine-tuning patent scope. Under this

148. O'Rourke, *supra* note 46; *cf.* Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 Stan. L. Rev. 1045 (1989) (suggesting that a program might lose all protection once it becomes an industry standard).

149. 17 U.S.C. § 906(a) (1994).

150. 17 U.S.C.A § 1201(f) (1996 & Supp. 2000).

151. Some have suggested to us that the reverse engineering right would be unnecessary if a patentee's disclosure obligation were sufficiently robust. While a real disclosure obligation (for example, a requirement to disclose source code) would ease the burden on competitors who want to know what is in a patented product, such a change in the law would not solve the problem altogether. Competitors still would not receive information about unpatented components of a computer program, and the only way to get access to that public domain information would be to reverse engineer the program as a whole.

152. *E.g.*, Markman v. Westview Instruments, Inc., 517 U.S. 370 (1996).

doctrine, a court may find infringement even where the accused product or process does not fall within the literal language of the patent claims, if it is nonetheless "substantially" equivalent to the patented invention. In essence, the doctrine targets conduct that violates the spirit, if not the letter, of the patent law—conduct that "should be" infringement even though, literally, it is not.[153] And since the doctrine of equivalents is no longer an equitable vehicle within the discretion of the trial judge, but a core part of every infringement case to be applied by the jury,[154] it is that broader doctrine that will effectively determine the scope of most litigated patents, regardless of how the claims are construed.[155]

At the same time, a central principle of the patent system is that the patentee is entitled to no more than she has claimed, and that the public is entitled to notice of the claims and what they encompass.[156] It follows that the criteria for finding equivalence are immensely important. Although commentators differ on exactly how the doctrine should be applied, they agree that courts interpreting the doctrine must walk a fine line between protecting innovators and stifling competition.[157]

---

153. *E.g.*, Graver Tank & Mfg. Co. v. Linde Air Prods. Co., 339 U.S. 605, 608 (1950) (observing that the doctrine exists "[to] temper unsparing logic and prevent an infringer from stealing the benefit of the invention") (quoting Royal Typewriter Co. v. Remington Rand, 168 F.2d 691, 692 (2d Cir. 1948)).

154. Warner-Jenkinson Co. v. Hilton Davis Chem. Co., 520 U.S. 17 (1997) (holding that the doctrine of equivalents is not equitable in nature, but should be applied as a matter of course, and strongly suggesting that equivalence is a question of fact rather than law).

155. Equivalents cases are particularly likely to go to the jury, since the Federal Circuit has held that defendants must meet a "lofty standard" to obtain summary judgment of noninfringement under the doctrine of equivalents, even where literal noninfringement is established as a matter of law. *See* Overhead Door Corp. v. Chamberlain Group, Inc., 194 F.3d 1261 (Fed. Cir. 1999).

156. *E.g.*, Sage Prods., Inc. v. Devon Indus., 126 F.3d 1420 (Fed. Cir. 1997):

> [A]s between the patentee who had a clear opportunity to negotiate broader claims but did not do so, and the public at large, it is the patentee who must bear the cost of its failure to seek protection for this foreseeable alteration of its claimed structure . . . . [T]he alternative rule—allowing broad play for the doctrine of equivalents to encompass foreseeable variations, not just of a claim element, but of a patent claim—also leads to higher costs. Society at large would bear these latter costs in the form of virtual foreclosure of competitive activity within the penumbra of each issued patent claim.

Id. at 1425; London v. Carson Pirie Scott & Co., 946 F.2d 1534, 1538 (Fed. Cir. 1991) ("[I]f the public comes to believe (or fear) that the language of patent claims can never be relied on . . . then claims will cease to serve their intended purpose. Competitors will never know whether their actions infringe a granted patent."); Laitram Corp. v. Cambridge Wire Cloth Co., 863 F.2d 855, 856-57 (Fed. Cir. 1988); Pennwalt Corp. v. Durand-Wayland, Inc., 833 F.2d 931, 935 (Fed. Cir. 1987). The notice principle is codified in 35 U.S.C. § 112 (1994).

157. *E.g.*, Merges & Nelson, *supra* note 51; John R. Thomas, *The Question Concerning Patent Law and Pioneer Inventions*, 10 High Tech. L.J. 35 (1995); Esther Steinhauer, Note, *Using the Doctrine of Equivalents to Provide Broad Protection for Pioneer Patents: Limited Protection for Improvement Patents*, 12 Pace L. Rev. 491 (1992); Timothy J. Douros, *Lending the Federal Circuit a Hand: An Economic Interpretation of the Doctrine of Equivalents*, 10 High Tech. L.J. 321 (1995). On the importance of clearly defining and limiting the doctrine of equivalents, see Hon. Paul R. Michel, *The Role and Responsibility of Patent Attorneys in Improving the Doctrine of Equivalents*, 40 Idea 123, 124 (2000).

As Part III.A explains, several characteristics of software suggest that the doctrine of equivalents is susceptible to especially generous application to software patents. Because software innovation typically involves considerable reuse of existing code, and because much of the innovation that occurs is not formally documented as prior art, software patents may be extended more broadly than patents on other inventions of comparable technical merit. Because software patents have a short effective life, the inclusion within a patent's scope of later-discovered equivalents, sanctioned by the Supreme Court's most recent interpretation of the doctrine,[158] will give holders of software patents control over many more generations of improvements than patentees in other industries. Finally, because software is embodied in text, triers of fact will need to select the appropriate level of abstraction at which to judge equivalence of function and must guard against overgeneralization. If courts fail to consider these factors, software patents, more than other types of patents, increasingly will act as broad "prospects" that reserve to the patentee the exclusive right to control innovation in related areas.[159] In Part III.B, we argue that this result is suboptimal given the nature of innovation within the software industry. Part III.C suggests doctrinal modifications, perhaps better described as interpretative canons, designed to allow courts to cabin software patent scope within appropriate bounds.

A. Systemic Biases Toward a Broad Range of Equivalents

The modern test for equivalence is set forth in the Supreme Court's opinion in *Warner-Jenkinson Co. v. Hilton Davis Chemical Co.*[160] Under the test, the patentee must demonstrate that the accused product or process incorporates the substantial equivalent of each element claimed in the patent.[161] It may do so by showing that the accused product or process has elements that perform "the same work in substantially the same way, and accomplish[es] substantially the same result" as each element in the patent claim.[162] Alternatively, it may introduce other evidence of equivalence, such as whether a "skilled practitioner" would have known of the interchangeability of the two elements at the time of the alleged infringement.[163]

---

158. *Warner-Jenkinson*, 520 U.S. at 37.

159. Kitch, *supra* note 47; *supra* Part I.C (discussing Kitch's prospect theory of optimal patent scope).

160. 520 U.S. 17 (1997).

161. *Id*. at 29 ("Each element contained in a patent claim is deemed material to defining the scope of the patented invention, and thus the doctrine of equivalents must be applied to individual elements of the claim, not to the invention as a whole.").

162. *Id*. at 35 (quoting Machine Co. v. Murphy, 97 U.S. 120, 125 (1878)). This test, given modern form in *Graver Tank & Manufacturing Co. v. Linde Air Products Co.*, 339 U.S. 605 (1950), has become known as the "triple identity" or "function-way-result" test. *See Warner-Jenkinson*, 520 U.S. at 39.

163. *Warner-Jenkinson*, 520 U.S. at 37-39.

In response, the defendant may offer evidence of noninterchangeability or show that the patentee surrendered its claim to the disputed technology during prosecution of the patent.[164] In *Warner-Jenkinson*, the Court let stand the Federal Circuit's determination that equivalence is a question of fact to be decided by the jury in cases where a jury has been requested.[165]

Of necessity, the *Warner-Jenkinson* analysis is highly fact-specific. "Substantial" equivalence will depend on the technological particulars of the case, as explicated by experts skilled in the relevant field. The Court also made clear that its list of relevant factors is illustrative, not exclusive.[166] In principle, therefore, there is also room to consider industry-specific factors that might bear on the question of the substantial equivalence or "known interchangeability" of particular components. In particular, we suggest that the following four factors are important in the software industry.

1. Incremental, Modular Innovation and Design for Interoperability

Assessments of equivalence depend, in part, on assessments of inventiveness. Both the scope of equivalence to be accorded the original invention and the latitude, if any, given the improvement depend on the degree of innovation (nonobviousness) in each product. On the one hand, a pioneering invention will be entitled to a broader range of equivalence than a more workmanlike one.[167] On the other, a pioneering improvement may be excused even from literal infringement under the "reverse doctrine of equivalents."[168] In between, an improvement that "designs around" an element of the patented invention will avoid infringement if the difference in the designs is substantial.[169] The doctrine's attempt to identify the requisite technical quantum of "designing around" reflects and promotes the patent law's utilitarian purpose; ultimately, "designing around" yields new technical paradigms, while simple imitation never does.

---

164. *Id*. at 35-37.

165. *See* Hilton Davis Chem. Co. v. Warner-Jenkinson Co., 62 F.3d 1512, 1521-22 (Fed. Cir. 1995) (en banc), *rev'd on other grounds*, 520 U.S. 17 (1997). While the Court granted certiorari to decide this issue, it concluded it was not necessary to do so. Nonetheless, the Court's reference to the use of special verdict forms to resolve doctrine of equivalence issues strongly suggests it views the jury as the appropriate decisionmaker in such cases.

166. *Warner-Jenkinson*, 520 U.S. at 39.

167. For a discussion of pioneering patents, see Steinhauer, *supra* note 157 and Thomas, *supra* note 157.

168. *E.g.*, Westinghouse v. Boyden Power Brake Co., 170 U.S. 537 (1898); Scripps Clinic & Research Found. v. Genentech, Inc., 927 F.2d 1565 (Fed. Cir. 1991); Merges, *Bargaining Breakdown*, *supra* note 81; Merges, *A Brief Note*, *supra* note 81.

169. The *Warner-Jenkinson* Court rejected the argument that the distinction between designing around and impermissible copying should turn on the second-comer's intent and instructed that judgments about equivalence must be based solely on objective, technical factors. *Warner-Jenkinson*, 520 U.S. at 34-36.

The distinction between designing around and mere imitation, though sensible on its face, is difficult to apply to software innovation because of the high degree of reuse that is standard operating practice. It is a truism that no patented invention is truly sui generis; each rests on what has gone before it.[170] This is particularly true of software-related inventions, however. Software innovation is by nature largely incremental.[171] It is rare for programs to be rewritten entirely from scratch; instead, innovation typically proceeds via a mix of new coding, modifications to some existing modules and subroutines, and either literal or functional reuse of others.[172] Moreover, patterns of improvements are constrained to a substantial degree by the need to preserve interoperability between program, system, and network components.[173]

The pattern of cumulative, sequential innovation and reuse that prevails in the software industry creates the risk that software patents will cast large shadows in infringement litigation. Specifically, we believe that because innovation is especially likely to proceed by building on existing code in other programs, the temptation for the trier of fact to find equivalence of improvements will be correspondingly greater. Put differently, most initial software inventions, although patentable, will not be pioneering advances entitled to a broad range of equivalence, simply because that is not the way that software innovation works. In addition, a jury asked to compare an accused computer program to a complex patent claim written in means-plus-function language may well be influenced by what it perceives as damning similarities between the two programs, even though it is only supposed to be considering similarities between the plaintiff's claim and the defendant's product. For both reasons, improvements that are real and substantial when judged against the background norms of the industry—norms of considerable and customary similarity—may be overlooked.

The analysis is similar for interoperability-related program elements. The need for interoperability does not preclude improvement, but it constrains the range of options available to the second-comer.[174] Here again, the temptation may simply be to find equivalence in the vast majority of cases because of surface similarities, without close consideration of the improvement's nature or its relation to the elements of the patented

---

170. On the implications of this cumulativeness for patent law generally, see Merges & Nelson, *supra* note 51, and Scotchmer, *supra* note 78, at 29.

171. Lemley & O'Brien, *supra* note 94; Samuelson et al., *supra* note 5; Menell, *supra* note 148.

172. Lemley & O'Brien, *supra* note 94; Samuelson et al., *supra* note 5. Literal reuse of code developed by other parties is, of course, prohibited by copyright law. Lemley & O'Brien, *supra* note 94; Samuelson et al., *supra* note 5. Functional reuse based on knowledge gained through reverse engineering, however, is not. *Supra* Part II.A.

173. Lemley & O'Brien, *supra* note 94; Samuelson et al., *supra* note 5; Lemley & McGowan, *supra* note 77.

174. *Cf*. Merges, *supra* note 46.

invention. Indeed, the interchangeability test may lead to pernicious results here, as computer programmers may prefer a particular improvement precisely because it is interchangeable with the original, even as they design around one or more of the original's features.[175]

It might seem odd to suggest that the standard for patent infringement be relaxed to accommodate an industry culture that favors incremental improvement. After all, the patent laws do not exist to encourage conservatism in design. Assuming the rightness of the basic utilitarian insight underlying grants of patent protection, insubstantial improvers should be forced to seek permission from and pay licensing fees to patentees, in the software industry, as elsewhere. We do not mean to argue that courts should jettison this basic model. We do, however, suggest that where software is concerned, first-cut judgments as to what is an insubstantial improvement may need to be rethought. This is not because software is special. Indeed, the same argument could be made in any industry in which sequential innovation plays a major role. Rather, it is because an improvement's importance can only be judged in the context of the art in which it occurs.

2. Undocumented Prior Art

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.[176] Innovations in biotechnology, for example, typically are documented in peer-reviewed professional journals, conference abstracts, and the like; software innovations, in contrast, may be documented only via developer specifications or online FAQs. Frequently, the source code itself is never released at all.[177] 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.[178]

---

175. On the equivalence of later-developed substitutes, see *infra* note 193.

176. Notice of Public Hearing and Request for Comments on Issues Related to the Identification of Prior Art During the Examination of a Patent Application, 64 Fed. Reg. 28,803 (May 27, 1999) [hereinafter *Prior Art Hearing Notice*]; Cohen, *supra* note 5, at 1178; Garfinkel, *supra* note 33, at 104.

177. *Supra* note 87 (discussing Federal Circuit's failure to require even software patentees to disclose source code); 37 C.F.R. § 202.20(c) (establishing special rules that exempt source code from Copyright Act's deposit requirements). The notable exception is the open source movement, whose members voluntarily release their source code. On the open source movement, see generally David McGowan, *Legal Implications of Open-Source Software*, 2000 U. Ill. L. Rev. (forthcoming 2000).

178. *E.g.*, *Prior Art Hearing Notice*, *supra* note 176; Cohen, *supra* note 5; Merges, *supra* note 5; Allan M. Soobert, *Breaking New Grounds in Administrative Revocation of U.S. Patents: A Proposition for Opposition—and Beyond*, 14 Santa Clara Computer & High Tech. L.J. 63 (1998); Nora M.

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 and indexing software-related developments defy even the most knowledgeable and diligent examiner.[179] 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. Congress has recently enacted other changes that would provide for publication of pending patent applications and would allow third parties to bring relevant prior art to the PTO's attention. These reforms, however, were watered down so much to satisfy opponents of patent reform that they offer no meaningful solution to the problem of software prior art.[180] 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.[181] 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.

The disconnect between the traditional patent examination process and software industry documentation practices has equally troubling

---

Tocups & Robert J. O'Connell, *Patent Protection for Computer Software*, 14 Computer Law 18 (Nov. 1997).

179. In 1994, the PTO revised its examiner credentialing requirements and began accepting examiner trainees with degrees in computer science. Cohen, *supra* note 5, at 1176. In addition, private parties created the Software Patent Institute, an organization designed to serve as a repository for software-related prior art, and began offering access and specialized training courses to the PTO. Garfinkel, *supra* note 33; Software Patent Institute, http://www.spi.org/ (last revised Dec. 17, 1999).

180. Patent reform was a long time in coming. *See* American Inventors Protection Act of 1999, H.R. 1907, 106th Cong. §§ 301-302, 402; Omnibus Patent Act of 1997, S. 507, 105th Cong. §§ 202, 302; 21st Century Patent System Improvement Act, H.R. 400, 105th Cong. §§ 202, 302; Omnibus Patent Act of 1996, S. 1961, 104th Cong. §§ 202, 302; Patent Reexamination Reform Act of 1995, S. 1070, 104th Cong. § 302; Patent Reexamination Reform Act of 1995, H.R. 1732, 104th Cong. § 302; Patent Application Publication Act of 1995, H.R. 1732, 104th Cong. § 2; Patent Reexamination Reform Act of 1994, S. 2341, 103d Cong. § 302; Patent Application Publication Act of 1994, S. 2488, 103d Cong. § 4. The bill that finally was passed, S. 1948, 106th Cong., 1st Sess., §§ 4001-4808, while it contains provisions for publication of pending applications and third-party participation in patent reexamination, is riddled with loopholes and limitations. For example, patentees can avoid publication of pending applications by promising not to file abroad. They can extend their patent beyond the 20 year term for any of a number of delays attributed to the PTO. And third-party reexamination is unlikely to be widely used because anyone who invokes it will be precluded from making similar arguments later in court.

181. Wayne M. Kennard, *Software Patents as a Weapon: Are You Ready to Rumble?*, 547 PLI/Pat. 1123 (1999). The PTO recently issued a call for public comments on ways to address deficiencies in the examination system. *Prior Art Hearing Notice*, *supra* note 176.

implications for infringement litigation. To invalidate an issued patent, an infringement defendant must overcome a strong presumption of validity.[182] If an infringement defendant loses a validity challenge, as most do,[183] the infringement analysis leaves little room for consideration of relevant but uncited prior art. For example, the rule of prosecution history estoppel, under which a defendant may escape a finding of infringement by showing that the patentee surrendered claim to the disputed material during prosecution, necessarily concerns only material of which the examiner had notice.[184] That art is often not the most relevant art available in litigation.[185]

This lack of consideration of uncited prior art, combined with the mode of analysis for determining equivalence, leaves little opportunity for courts to constrain the scope of patents under the doctrine of equivalence. The element-by-element approach to equivalence, while properly cabining the jury's power to expand the scope of a patent, may not help in many software cases, where the software-related part of the invention is often described in a single element.[186] Moreover, the "known interchangeability" rule outlined by *Warner-Jenkinson* sweeps within the patent's scope any material known to be substantially equivalent, without consideration of whether the material, if cited during prosecution, would have required narrowing of the claims prior to issuance.[187] The Federal Circuit's *Wilson Sporting Goods* opinion seemed to require such consideration, but in subsequent opinions the court has not consistently required this hypothetical inquiry and has left the burden of proof on matters relating to uncited prior art unclear.[188]

---

182. 35 U.S.C. § 282 (1994). *See* Mark A. Lemley, *Rational Ignorance at the Patent Office*, 95 Nw. U. L. Rev. (forthcoming 2001). Relaxing this presumption for software patents might seem an obvious solution. *But see* Bausch & Lomb, Inc. v. Alcon Labs., Inc., 79 F. Supp. 2d 252 (W.D.N.Y. 2000) (excluding testimony about inefficiencies in PTO's examining system, proffered to support argument that presumption should be relaxed); Applied Materials, Inc. v. Advanced Semiconductor Materials Am., Inc., 32 U.S.P.Q.2d 1865 (N.D. Cal. 1995) (same).

183. John R. Allison & Mark A. Lemley, *Empirical Evidence On The Validity Of Litigated Patents*, 26 AIPLA Q.J. 185, 205-06 (1998) (reporting that 54% of all patents litigated are found valid).

184. Warner-Jenkinson Co. v. Hilton Davis Chem. Co., 520 U.S. 17, 32-33 (1997).

185. Allison & Lemley, *supra* note 183, at 233 (noting that validity challenges are more successful when based on art not cited by the PTO).

186. Thus, in *Overhead Door Corp. v. Chamberlain Group, Inc.*, 194 F.3d 1261 (Fed. Cir. 1999), the court held that a mechanical switch could be equivalent to an electronic switch implemented in software. Had the switch itself been claimed in terms of its parts, it is doubtful an electronic switch could have an equivalent to each part. But because the switch was itself only a single element in a broader claim to a garage door opener, substituting software for a mechanical device was held potentially equivalent.

187. *Warner-Jenkinson*, 520 U.S. at 36-39.

188. Wilson Sporting Goods Co. v. David Geoffrey & Assocs., 904 F.2d 677, 684-85 (Fed. Cir. 1990):

> [S]ince prior art always limits what an inventor could have claimed, it limits the range of permissible equivalents of a claim . . . . [I]t may be helpful to . . . visualiz[e] a *hypothetical* patent claim, sufficient in scope to *literally* cover the accused product. The pertinent question

We suspect that the doctrine of equivalents' inadequate recognition of uncited prior art may combine with the highly incremental character of software innovation to produce a broad "umbrella effect" for issued software patents. As discussed above, industry-specific patterns of cumulative innovation suggest that an improvement upon patented software technology is more likely to be deemed within the patent's range of equivalents. At the same time, industry-specific patterns of documentation increase the likelihood that the original patent will be too broad, incorporating unsung but vital improvements that preceded and should have narrowed it. This is an odd result for a doctrine that is fundamentally equitable in purpose. The *Warner-Jenkinson* Court stated that the doctrine is not, as a technical matter, a rule of equity, and rejected the use of some traditionally equitable factors, such as intent, in equivalence cases. At the same time, however, the Court also reaffirmed the doctrine's basic aim: to guard against the excesses of literalism in claim construction, while preserving the essence of the public's right to notice.[189] In short, the doctrine of equivalents is a shield, not a sword. It exists to preserve the patentee's rights to her own invention, not to give the patentee more than she has actually invented.[190] Allowing software patentees to claim a broad range of equivalents because of industry-specific defects in the prosecution system puts the cart before the horse.

3. The Rapid Pace of Change

As noted above, *Warner-Jenkinson* requires assessment of the "known interchangeability" of an accused improvement based on a reasonably skilled practitioner's knowledge at the time of alleged infringement.[191] In this respect, *Warner-Jenkinson* arguably changes the law; although some Federal Circuit decisions had interpreted the doctrine to cover later-discovered equivalents, language in earlier Supreme Court opinions suggested knowledge at the time of invention as the benchmark for equivalence determinations.[192] As commentators have recognized, this

---

then becomes whether that hypothetical claim could have been allowed by the PTO over the prior art.

Id. Compare Ultra-Tex Surfaces, Inc. v. Hill Bros. Chem. Co., 204 F.3d 1360 (Fed. Cir. 2000), *and* Streamfeeder, LLC v. Sure-Feed Sys., Inc., 175 F.3d 974 (Fed. Cir. 1999) (both endorsing the hypothetical claim analysis) *with* Nat'l Presto Indus., Inc. v. West Bend Co., 76 F.3d 1185 (Fed. Cir. 1996), *and* Conroy v. Reebok Int'l, Ltd., 14 F.3d 1570 (Fed. Cir. 1994) (both seeming to restrict its use).

189. *Warner-Jenkinson*, 520 U.S. at 29. ("There can be no denying that the doctrine of equivalents, when applied broadly, conflicts with the definitional and public-notice functions of the statutory claiming requirement.").

190. Graver Tank & Mfg. Co. v. Linde Air Prods. Co., 339 U.S. 605, 608 (1950).

191. *Warner-Jenkinson*, 520 U.S. at 37.

192. *Compare, e.g.*, Hughes Aircraft Co. v. United States, 717 F.2d 1351 (Fed. Cir. 1983) *with Graver Tank & Mfg. Co.*, 339 U.S. at 609 ("An important factor is whether persons reasonably skilled

formulation of the test ensures that the scope of an issued patent will expand to keep pace with later-discovered variations on the basic technology.[193] However, it also allows a patent to encompass even currently known products that are later discovered to be "interchangeable" with an element of the patented invention.

From a theoretical standpoint, the time-of-infringement test replaces the doctrine's former, more recognizably equitable focus on bad faith with a broader, more pragmatic focus on the recovery of sunk costs.[194] Put simply, the test insures patentees against the vagaries of the after-market for improvements. This has the virtue of eliminating the "20-20 hindsight" problem; it is far easier for an expert to say what she thinks of an improvement today than what she would have thought of it had it been made four or more years ago. Yet by collapsing the infringement inquiry into a single timeframe, the test may underestimate the full extent of the second-comer's improvement. Indeed, we suspect that the time of infringement test systematically undervalues the significance of subsequent improvements, for the same reason that hindsight often leads observers to label obvious in retrospect an invention that was significant at the time it was made.

Although the time-of-infringement test ensures that the scope of any patent will widen throughout its life, the test is likely to produce especially strong effects for software patents. The effective life of a software innovation is normally quite short, much shorter than the nearly twenty-year term conferred by patent law.[195] In many other fields, a patented invention will be marketable for the full patent term; software innovations rarely demonstrate the same sort of staying power. Accordingly, ex post expansions in patent scope may be expected to yield proportionately greater increases in profitability for software patents than for other types of patents.[196]

More important, over twenty years, a software patent expanded to cover later-discovered improvements will exert control over many more generations of improvements than a conventional patent with a longer effective term, at least if the patent is read under the doctrine of equivalents to encompass more than the specific way in which it has been

---

in the art would have known of the interchangeability of an ingredient not contained in the patent with one that was."), *and* Halliburton Oil Well Cementing Co. v. Walker, 329 U.S. 1, 13 (1946).

193. Robert P. Merges et al., Intellectual Property in the New Technological Age 279-82 (2d ed. 2000); James R. Farrand & Ronald R. Johnston, *Expanded Doctrine of Equivalents Extends Patents Old and New*, 14 Computer Law. 1 (1997).

194. In this respect, the rule is consistent with the Court's insistence that the doctrine of equivalents is utilitarian, rather than strictly equitable, in purpose. *Warner-Jenkinson*, 520 U.S. at 34-35.

195. Samuelson et al., *supra* note 5, at 2431 n.134.

196. *See* Suzanne Scotchmer, *Cumulative Innovation in Theory and Practice* (U.C. Berkeley Goldman School of Public Policy, Working Paper No. 240, Feb. 1999) (on file with Julie E. Cohen) (identifying effective patent life as an important determinant of patent profitability).

implemented.[197] This means that, practically speaking, the market-distorting effect of a software patent—in economic terms, the "deadweight loss" imposed on society—will be substantially greater than for other types of patents.

Arguably, allowing software patentees to capture the value of improvements many generations removed from the initial invention simply preserves incentives to innovate in the face of rapid technological change. Within the traditional patent law framework, however, the desire to preserve incentives coexists with other doctrines, including the reverse doctrine of equivalents, designed to ensure that issued patents do not cut too deep a generational swath.[198] At a minimum, then, we should inquire whether this approach also produces correspondingly greater social benefits.[199]

4. Equivalence and Text

Finally, judging the equivalence of software-related innovations presents difficulties because of the medium in which these innovations are embodied. Although software in usable form exists as a series of electronic impulses, the medium of software innovations—the medium in which the innovative activity occurs—is text.[200] Code-based innovation is, of course, constrained to a significant degree by the formal dictates of logic. Nonetheless, software innovations have a degree of plasticity that other innovations lack.

As at least one commentator has observed, evaluating code for equivalence presents problems conceptually similar to those entailed in judging originality and substantial similarity in copyright.[201] Before the accused program can be compared to a patent claim, two steps must occur. First, the court must interpret the claim. This step does not require parsing of code; software patents are not normally claimed or defined in terms of the actual code used by the patentee.[202] Rather, the technological advance

---

197. Courts have actually differed on whether the doctrine of equivalents extends software patents across generations. For a discussion of specific cases, see *infra* notes 229-237 and accompanying text.

198. *E.g.*, Westinghouse v. Boyden Power Brake Co., 170 U.S. 537 (1898); Scripps Clinic & Research Found. v. Genentech, Inc., 927 F.2d 1565 (Fed. Cir. 1991); Merges, *A Brief Note*, *supra* note 81; *see also* Texas Instruments, Inc. v. U.S. Int'l Trade Comm'n, 846 F.2d 1369 (Fed. Cir. 1988) (affirming finding that means-plus-function claims were not literally infringed, despite presence in accused device of functions corresponding to each element of the claims, because totality of technological improvement in accused device rendered device nonequivalent).

199. For further discussion of this point, see *infra* Part III.B.

200. Samuelson et al., *supra* note 5, at 2320-26.

201. David A. Shough, *Infringement of Hardware Patents by Software-Controlled Devices: A Study of Equivalence*, 8 J. Proprietary Rts. 8 (1996); *cf.* Peterson, *supra* note 43 (arguing that evaluating code-based innovations for nonobviousness raises similar problems); Durham, *supra* note 23, at 1522-26 (same).

202. A possible exception is means-plus-function claim language, for which claim interpretation must refer back to the "structure" disclosed in the patent specification. 35 U.S.C. § 112 ¶ 6 (1994).

embodied in the code is described in the claim; interpretation proceeds according to standard canons of claim construction. Because all patents are ultimately defined by text, this linguistic problem exists for all kinds of patents.[203] A patent claim that is written at a higher conceptual level will be interpreted differently than one written with more concrete detail. The problem is aggravated in the case of software patents, however. Many software patents, especially first-generation ones, give little or no information in the patent claim (or indeed in the specification) about the software program itself.[204] Even a later-generation software patent claim may tell the court very little about the software program in question, leading to greater variance in the level of abstraction selected.[205] Software is in certain respects more malleable than many other types of inventions (such as pharmaceuticals or mechanical devices). Two pieces of code may produce the same result and may even use very similar algorithms to do so, but may still operate differently, for example, by extracting output data from a memory array in a different manner.[206]

Second, the factfinder must determine the appropriate conceptual level at which the accused device or process will be viewed for purposes of comparison to the claim as interpreted by the court. At this stage, whether an accused program satisfies a disputed element of the patented invention may depend entirely on the chosen level of abstraction, for the same reason just described.[207] Evaluation of the accused program is further complicated

---

Whether this "structure" includes the computer program itself, or merely the physical substrate in which it is embodied, is a contested issue. *See* WMS Gaming, Inc. v. Int'l Game Tech., 184 F.3d 1339 (Fed. Cir. 1999) (holding that an algorithm is part of the structure for literal infringement, but not necessarily under the doctrine of equivalents).

203. *E.g.*, Autogiro Co. of Am. v. United States, 384 F.2d 391, 396 (Ct. Cl. 1967) ("The very nature of words would make a clear and unambiguous [patent] claim a rare occurrence."); Craig Allen Nard, *Certainty, Fence Building, and the Useful Arts*, 74 Ind. L.J. 759, 760 (1999); John R. Thomas, *On Preparatory Texts and Proprietary Technologies: The Place of Prosecution Histories in Patent Claim Interpretation*, 47 UCLA L. Rev. 183 (1999).

204. Often this is because patent drafters were trying to conceal the very fact that they were patenting software, back in the days when you had to pretend you were doing something else.

205. *Cf. supra* note 87 (discussing Federal Circuit's virtual elimination of the enablement and best mode requirements for software patent claims). Here again, means-plus-function claims may be an exception. *Supra* note 202. These problems, of course, also complicate initial decisions about software patent issuance. In at least some cases, the plasticity of software and lack of supporting detail required, along with the prior art problems discussed above, *see supra* Part III.A.2, will lead to the allowance of claims that are too broad, with obvious ramifications for infringement analysis. *Cf.* Peterson, *supra* note 43.

206. *See* Wiener v. NEC Electronics, Inc., 102 F.3d 534 (Fed. Cir. 1996); Alpex Computer Corp. v. Nintendo Co., 102 F.3d 1214 (Fed. Cir. 1996).

207. *See* Shough, *supra* note 201, at 13-17; *cf.* MiTek Holdings, Inc. v. Arce Eng'g Co., 89 F.3d 1548, 1559 (11th Cir. 1996); Bateman v. Mnemonics, Inc., 79 F.3d 1532, 1544-45 (11th Cir. 1996); Apple Computer, Inc., v. Microsoft Corp., 35 F.3d 1435 (9th Cir. 1994); Eng'g Dynamics, Inc. v. Structural Software, Inc., 26 F.3d 1335, 1342-43 (5th Cir. 1994); Gates Rubber Co. v. Bando Chem. Indus., 9 F.3d 823, 834 (10th Cir. 1993); Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992); *cf. also* Brown Bag Software v. Symantec Corp., 960 F.2d 1465 (9th Cir. 1992); Mark A. Lemley, *Convergence in the Law of Software Copyright?*, 10 High Tech. L.J. 1 (1995). Courts have,

by the fact that compiled code may perform steps in a different order than the written source code might suggest, a fact that may matter depending upon the court's interpretation of what the patent claim requires.

An example of how the level of abstraction selected at both stages can influence the outcome of the doctrine of equivalents analysis is *Overhead Door Corp. v. Chamberlain Group, Inc.*[208] In that case, the Federal Circuit held that a reasonable jury could find that a patent for a garage door opener using a mechanical switch was equivalent to an accused device that used an electronic switch implemented in software. The result follows from the court's implicit decision to interpret the switch element at a high level of abstraction—that is, to believe that the "way" in which it functioned was by turning on or off. By contrast, an analysis at a lower level of abstraction, one that inquired into how the claimed and accused switches actually worked, would surely have found substantial differences between a physical lever and a computer program.[209] In sum, just as in copyright cases comparison at an overly general level of abstraction will tend to yield a finding of infringing similarity, so too with software patent cases.[210]

The Federal Circuit's recent decisions on equivalence claims in software cases indicate an awareness of the need to find equivalence at the level of detailed program structure as well as function.[211] The use of juries rather than judges to decide questions of equivalence complicates the matter still further, however.[212] Courts in copyright cases have recognized that

---

upon occasion, used the "reverse doctrine of equivalents" to avoid a finding of equivalence in cases presenting substantial technological improvement in the performance of a given function, *infra* notes 228-236 and accompanying text, but that doctrine is neither intended nor well tailored to address routine errors caused by inexact and overly general analysis of the accused program, or of the patent claim.

208. 194 F.3d 1261 (Fed. Cir. 1999).

209. This problem is not new. It has reared its head in various guises throughout the history of patent law, most recently as courts try to determine what an "element" is for purposes of the all-elements rule endorsed in *Warner-Jenkinson*. For milestones in this sub rosa debate, see, for example, *Hughes Aircraft Co. v. United States*, 140 F.3d 1470 (Fed. Cir. 1998), and *Corning Glass Works v. Sumitomo Electric U.S.A., Inc.*, 868 F.2d 1251 (Fed. Cir. 1989).

210. The classic statement of the problem is Learned Hand's:

> Upon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may perhaps be no more than the most general statement of what the play is about, and at times may consist only of its title; but there is a point in this series of abstractions where they are no longer protected, since otherwise the playwright could prevent the use of his 'ideas' . . . .

Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (2d Cir. 1930).

211. *See* General Elec. Co. v. Nintendo Co., 179 F.3d 1350 (Fed. Cir. 1999) (affirming summary judgment of noninfringement where accused device used substantially different structure to produce equivalent result); Digital Biometrics, Inc. v. Identix, Inc., 149 F.3d 1335 (Fed. Cir. 1998) (same); *Wiener*, 102 F.3d at 1023 (same); *Alpex Computer Corp.*, 102 F.3d at 1214 (reversing judgment of infringement based on "equivalence of functional result" where evidence showed that accused device was substantially different in structure and operation).

212. *See* Warner-Jenkinson Co. v. Hilton Davis Chem. Co., 520 U.S. 17, 37-39 (1997).

selection of the appropriate level of abstraction may be influenced by a variety of factors unrelated to technical considerations, including unfamiliarity with the subject matter of the dispute and misunderstanding of the degree to which the law allows similarity between works. Accordingly, they routinely instruct jurors on the difference between copyrightable expression and uncopyrightable ideas and methods of operation, and have modified the traditional "lay observer" test for substantial similarity to allow consideration of expert testimony where computer software is involved.[213] Juries in patent cases, of course, already receive considerable expert guidance, but it is not ordinary practice to instruct juries in patent cases on levels of abstraction. Instead, in patent cases, it is the court's duty to tell the jury what the claims mean.[214] It seems, therefore, that it also should be the court's duty to identify the appropriate level of abstraction at which the jury should compare the patented invention and the improvement under the doctrine of equivalents.

B. Innovation and Equivalence: An Industry-Based Analysis

The doctrine of equivalents seeks to fine tune the patent system's ability to address its central task: ensuring sufficient rewards (and therefore sufficient incentives) to patentees while avoiding an unnecessary degree of deadweight loss to society as a whole. The conventional wisdom is that, at least as a general matter, the deadweight losses imposed by the existence of the patent system are worth it. As Part I.A discussed, the conventional wisdom has arrived at the identical conclusion with respect to the specific question of patent protection for software-related inventions.[215] This Article does not challenge either answer, but asks instead whether, given the choice to award patents for software-related inventions, the incentive-deadweight loss tradeoff flowing from that choice is optimized by a broad or narrow approach to patent scope.

The policy question presented can be described using the following matrix:

---

213. *E.g.*, Apple Computer, Inc. v. Microsoft Corp., 35 F.3d 1435, 1442-43, 1445 (9th Cir. 1994); Gates Rubber Co. v. Bando Chem. Indus., 9 F.3d 823, 834 (10th Cir. 1993); Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 712-13 (2d Cir. 1992).

214. Markman v. Westview Instruments, Inc., 517 U.S. 370 (1996).

215. *E.g.*, State Street Bank & Trust Co. v. Signature Fin. Group, Inc., 149 F.3d 1368 (Fed. Cir. 1998); *In re* Alappat, 33 F.3d 1526 (Fed. Cir. 1994) (en banc); Scott M. Alter, *Federal Circuit Broadens Scope for Software Patents*, 15 Computer Law. 24 (Oct. 1998); Wesley L. Austin, *Software Patents*, 7 Tex. Intell. Prop. L.J. 225 (1999); Vincent Chiapetta, *Patentability of Computer Software Instruction as an "Article of Manufacture:" Software as Such as the Right Stuff*, 17 J. Marshall J. Computer & Info. L. 89 (1998). *But see* Kreiss, *supra* note 43, at 31; Thomas, *supra* note 97.

	Improvement Patentable	Improvement Not Patentable
Improvement Within Original Patent Scope	A. Blocking Patents (or reverse doctrine of equivalents)	B. Infringement
Improvement Outside Original Patent Scope	C. Standalone Patent (new invention sequence)	D. Routine Cumulative Innovation

According initial software patents a broad scope shifts some patentable improvements from Box *C* to Box *A*, and shifts some subpatentable improvements from Box *D* to Box *B*. Whether this is good policy depends on the relative importance of inventions in these Boxes to patterns of innovation within the software industry as a whole. The patent literature traditionally has answered this question by focusing on the bargaining abilities of improvers who receive "blocking patents," and arguing that the system encourages (or, with slight modifications, will encourage) inventors and improvers who hold blocking patents to bargain to mutually acceptable results.[216] It should be apparent, however, that our concern is not only or even primarily with the occupants of Box *C*, for precisely that reason. The bargaining position conferred by a blocking patent means that the occupants of Box *C*, if shifted to Box *A*, have at least some mechanisms available to protect themselves.[217] Our focus, instead, should be the occupants of Box *D*, those whose improvements are not patentable in their own right, who would be deemed simple infringers under a regime of broad equivalence, and who represent the vast majority of cases. What is to be gained by making infringers out of these routine innovators?

One justification for the shift, perhaps, is improved cost recovery for patentees. As Scotchmer notes, the effective term of a software patent is

216. Under the patent law, an improver may receive a patent on an infringing improvement. Such patents are called "blocking patents" because they block the original patentee from practicing the improvement unless licensed to do so. *E.g.*, Lemley, *supra* note 51 (arguing that the blocking patents system promotes improvement to a greater degree than the copyright system, even though it does not afford complete insurance against bargaining breakdown); Merges, *supra* note 168 (arguing that the reverse doctrine of equivalents should be strengthened to enhance the bargaining position of improvers who hold blocking patents); Scotchmer, *supra* note 196.

217. For a discussion of these mechanisms, as well as their shortcomings, see the sources cited *supra* note 199.

very short, which creates incentive problems for would-be developers of patentable inventions.[218] But all software innovations have short effective lives, whether patentable or not. Allowing initial patents a broad scope simply shifts costs and a great deal of risk to follow-on innovators (Box *D*) who face equal time pressure.

Shifting costs and risk is, of course, part of the point of the patent system; those who prefer certainty to risk may bargain for it.[219] Thus, a second, more pragmatic justification for according initial patents a broad scope is expansion of the patentee's licensing pool. Indeed, some commentators argue that both deadweight-loss and cost-recovery concerns would be addressed most effectively under a collective-rights framework involving all members of the affected industry.[220] As an argument for broad patents, however, the "bargaining pool" reasoning proves too much. Bargains and collective agreements may be based on broad patents or on narrower ones. The premise of collective rights systems, after all, is that participants will wish to purchase certainty regardless of baseline legal entitlements.[221] Baseline entitlements matter for a different reason: they affect both initial bargaining positions and final outcomes.[222]

Indeed, a focus on collective-rights solutions suggests that broadening patent scope is the wrong approach for promoting progress within the software industry. From the standpoint of both current and would-be participants, certainty is not the only significant feature of a collective-rights model. It is also important that not only established patent holders but also newcomers and small stakeholders have continued incentives to innovate. This is particularly so in an industry with many players and a constant

---

218. Scotchmer, *supra* note 196.

219. *See id.*

220. Robert P. Merges, *Contracting Into Liability Rules: Intellectual Property Rights and Collective Rights Organizations*, 84 Calif. L. Rev. 1293 (1996); Scotchmer, *supra* note 196. For similar reasons, other commentators have recommended simply excluding software from the traditional property-rule framework of patent law and subjecting it to a regime of rights based on liability rules. *See* J.H. Reichman, *Solving the Green Tulips Problem: Packaging Rights in Subpatentable Innovation*, 53 Vand. L. Rev. (forthcoming 2000); Samuelson et al., *supra* note 5; *cf.* J.H. Reichman & Pamela Samuelson, *Intellectual Property Rights in Data?*, 50 Vand. L. Rev. 51 (1997) (advocating a similar regime for databases). Under such a regime, software developers would surrender their rights to exclude in exchange for guaranteed fees from users and improvers. Newcomers and small-timers, meanwhile, would simply purchase licenses, at predetermined rates, to use desired innovations.

221. Merges has argued that granting property rights may encourage the development of collective rights organizations that efficiently allocate rights between licensors and licensees, and that those organizations may create their own liability rule regimes by contract. Merges, *supra* note 220. But in the cases we are describing here, the question is not whether to grant property rights to inventors, but which of two different inventors (the initial inventor or the improver) should hold a particular property right. In that situation, Merges has noted the importance of dividing entitlements. Merges, *Bargaining Breakdown*, *supra* note 81.

222. Merges, *Bargaining Breakdown*, *supra* note 81 (advocating a strengthened bargaining position for improvers, for precisely this reason).

supply of new entrants.[223] The software industry has precisely these characteristics. Indeed, because of the many generations of improvers who would have to bargain with an initial inventor, it may be unrealistic to think that most or even many efficient transactions will occur.

More generally, the presumption that only pioneering improvers are worth protecting is inappropriate for an industry characterized by networked, interdependent products, and protecting only pioneering improvers will have the effect of encouraging moves from Boxes *D* and *B* to Boxes *C* and *A*; that is, encouraging industry participants to make larger rather than smaller changes to existing programs. The resulting pattern of innovation by leaps and bounds (rather than incremental innovation) may actually decrease social welfare, both by reducing interoperability among programs (and therefore foregoing the corresponding network benefits) and by rendering the resulting untested programs less reliable. If so, treating software patents as broad "prospects" will hinder progress.

C. Tailoring the Doctrine of Equivalents to the Software Industry

In sum, there are strong policy reasons for avoiding a broad "prospect" approach to software patent scope, but there is significant likelihood that courts (and juries) will take this approach, perhaps inadvertently. To avoid this danger, courts should develop a set of interpretative canons for assessing equivalence in software patent cases that takes into account the industry-specific factors described in Part III.A, above.

The first and fourth factors are precautionary. They simply require courts to consider the appropriate level of abstraction[224] and factor in the background norm of incrementalism[225] when construing claims and instructing juries. Juries, in turn, must be given the proper frame of reference for comparison, and for assessing the degree of variance between invention and improvement.

The second and third factors require greater doctrinal adjustment. We believe, however, that these adjustments can be accomplished within the doctrine of equivalents. They do not require special treatment for software

---

223. *Cf.* Elinor Ostrom, Governing the Commons: The Evolution of Institutions for Collective Action 205-07, 211 (1990) (observing that what works for small, stable communities will not necessarily work when community membership becomes more fluid). Grossly unequal treatment of newcomers and small stakeholders was an important factor in the U.S. government's decision to sue the two major performing rights societies, ASCAP and BMI, for antitrust violations, and the resulting consent degrees were substantially shaped by fairness concerns. United States v. Broadcast Music, Inc., 1966 Trade Cas. (CCH) ¶ 71,941 (S.D.N.Y. 1966), *as amended*, 1996-1 Trade Cas. (CCH) ¶ 71,378 (S.D.N.Y. 1994); United States v. Am. Soc'y of Composers, Authors & Publishers, 1950 Trade Cas. (CCH) ¶ 62,595 (S.D.N.Y. 1950); United States v. Am. Soc'y of Composers, Authors & Publishers, 1941 Trade Cas. (CCH) ¶ 56,104 (S.D.N.Y. 1941); John Ryan, The Production of Culture in the Music Industry 92-100 (1985).

224. *Supra* text accompanying notes 205-210.

225. *Supra* text accompanying notes 171-175.

patents, just detailed attention to problems that occur with particular frequency in the software industry. In light of industry and institutional barriers to comprehensive prior art searches,[226] it seems reasonable to suggest that the "known interchangeability" standard be modified in cases involving computer software. Specifically, when a court rejects a validity challenge based on uncited prior art of the sort we describe here, it should nonetheless inquire whether, if the reference had been cited to the examiner, the patent would have been narrowed in a way that would save the accused improvement. The doctrinal basis for this adjustment can be found in a proper application of *Wilson Sporting Goods*.[227]

The generational distortions caused by the time-of-infringement test,[228] meanwhile, can be addressed using a variant of the reverse doctrine of equivalents. Just as that doctrine excuses literal infringement if the improvement is pioneering in its own right, so it can and should excuse equivalent infringement if the improvement is so many generations removed from the knowledge that produced the original invention that it constitutes a substantial departure from the original.[229]

There are some encouraging signs for our approach in the software infringement cases recently decided by the Federal Circuit. Most of these decisions have rejected arguments that read claim language written for one product generation at such a high level of abstraction that it covers accused products from a different generation. Thus, in *Alpex Computer Corp. v. Nintendo Co.*,[230] the Federal Circuit held that a patent claim to a video game output display system was not infringed by a next-generation system that worked in a different way. Alpex's claimed system included a display RAM that stored information corresponding to each pixel of a television screen in a discrete location. Nintendo's accused device, by contrast, used shift registers to store one "slice" of the video display at any given time. The Federal Circuit rejected a jury finding that the two systems were equivalent.[231] In *Digital Biometrics, Inc. v. Identix, Inc.*,[232] the court construed narrowly a patent claim to "image arrays" storing a two-dimensional slice of video data, and which were merged into a "composite array"

226. *Supra* text accompanying notes 176-181.

227. *Supra* note 188 and accompanying text.

228. *Supra* text accompanying notes 191-199.

229. For an example of such qualitative, generational improvement, see Andrew Chin, *Computational Complexity and the Scope of Software Patents*, 39 Jurimetrics J. 17 (1998).

230. 102 F.3d 1214 (Fed. Cir. 1996).

231. *Id.* at 1222. *Wiener v. NEC Electric, Inc.*, 102 F.3d 534 (Fed. Cir. 1996) is similar to *Alpex*. In *Wiener*, the Federal Circuit upheld the district court's finding of noninfringement under the doctrine of equivalents, because there were substantial differences between the patent's requirement that a computer program "call on" columns of data one byte at a time and the defendant's product, in which the columns alleged to be equivalent were not in the data matrix, and therefore were not called upon to read data. The court rejected the "conclusory" declaration of plaintiff's expert that the two processes were identical. *Id.* at 542.

232. 149 F.3d 1335 (Fed. Cir. 1998).

storing a fingerprint image. The court held that the defendant's systems, which constructed the composite array directly rather than by using two-dimensional slices, did not create "image arrays" within the meaning of the claims.[233] Most recently, in *Wang Laboratories, Inc. v. America Online*,[234] the court affirmed a district court decision granting summary judgment of noninfringement under the doctrine of equivalents. The patent claims in that case covered "frames," defined in the specification as pages encoded in character-based protocols. The court rejected Wang's attempt to extend the patent to cover bit-mapped pages, crediting evidence that there were "huge, huge differences" between the two approaches.[235]

Other cases, however, have applied the doctrine of equivalents more broadly. In some of those cases, the Federal Circuit has found equivalence between two different types of software programs written in different product generations. *WMS Gaming, Inc. v. International Game Technology*[236] is instructive. In that case, the court held that a claim written in means-plus-function language that relied for its corresponding structure on a computer programmed with a particular algorithm was limited in literal scope to the particular algorithm chosen and equivalents thereof. However, the court found the defendant's algorithm infringing under the doctrine of equivalents. This latter approach has the potential to expand the scope of patents in the software industry dramatically.[237] More troubling, some cases suggest that software implementations of certain ideas are equivalent to older mechanical implementations. An example is *Overhead Door Corp. v. Chamberlain Group, Inc.*,[238] discussed above.[239] The patented system claimed a (mechanical) switch connected to a microprocessor, which could store the codes of multiple garage doors. The Federal Circuit held that the claim was not literally infringed by an electronic switch implemented in software. However, the court reversed a grant of

---

233. *Id.* at 1349.

234. 197 F.3d 1377 (Fed. Cir. 1999).

235. *Id.* at 1386. In a related context (interpreting equivalent structure in a means-plus-function claim), in *General Electric Co. v. Nintendo Co*., 179 F.3d 1350 (Fed. Cir. 1999), the court held that Nintendo's video game systems did not infringe GE's television switch patents because the patents, written in means-plus-function format, did not disclose a function for the switches identical to Nintendo's function. On the differences between the doctrine of equivalence and equivalence under a means-plus-function analysis, see *Chiuminatta Concrete Concepts, Inc. v. Cardinal Industries*, 145 F.3d 1303 (Fed. Cir. 1998).

236. 184 F.3d 1339 (Fed. Cir. 1999).

237. It also suggests, however, that software patents that merely implement a well-known mechanical concept in software may be vulnerable to an obviousness challenge. This follows from the *Wilson Sporting Goods* rule, coupled with the interrelatedness of claim construction. *Cf.* Donald S. Chisum, *Anticipation, Enablement and Obviousness: An Eternal Golden Braid*, 15 AIPLA Q.J. 57 (1987) (discussing the latter). If mechanical claim elements are properly viewed as equivalent to a computer program that performs the same function, it may well follow that the computer program is obvious in view of the mechanical prior art.

238. 194 F.3d 1261 (Fed. Cir. 1999).

239. *Supra* Part III.A.4.

summary judgment to the defendants under the doctrine of equivalents, concluding that a reasonable jury could find that the difference between mechanical and software implementations was a mere "design choice."[240]

These troubling inconsistencies in the Federal Circuit's software patent decisions indicate that a more systematic approach to questions of software patent scope is needed. We believe that the set of interpretative canons that we have identified will produce greater consistency, and will provide better guidance to the federal district judges who must try patent cases and instruct juries. The result will be a body of decisional law that is more predictable, and that more effectively promotes innovation by software firms.

Conclusion

Exploration of the consequences of patent protection for innovation in the software industry is just beginning. Here, we have tried to suggest some of the pitfalls that existing patent doctrine may create for software developers throughout the research and development process. Because software must be reverse engineered to be understood, the patent law's failure to provide a reverse engineering privilege may pose unique difficulties for software research, and thus may frustrate fundamental patent polices favoring disclosure and competition. Because software innovations tend to be incremental and poorly documented, and because their economic lives tend to be much shorter than the uniform patent term, courts may apply the doctrine of equivalents too broadly in software infringement disputes, and thus may stifle efforts by second-comers to design around existing patents. Further, these problems are linked. Robust competition by improvers requires both that they be able to engage in reverse engineering in order to analyze existing programs, and that they have the freedom to design new products without undue risk of liability for patent infringement.

In short, in both of the situations we have identified, applying existing patent doctrine to software patents threatens to create exclusionary rights that are extraordinarily broad even by patent standards. To a substantial degree, this would accord with the requirements of a "prospect" approach to software patents. As we have shown, however, that result is unlikely to promote progress in this industry. Because of the unique technical and economic characteristics of software, patent protection that is broader than usual is much more likely to hinder innovation than to foster it.

Our proposal is essentially a conservative one: In extending the full benefits of patent protection to software, courts must make sure that the unique characteristics of software do not result in an unprecedented and equally unique expansion of patent scope. To help courts or Congress

240. *Overhead Door Corp.*, 194 F.3d at 1270.

achieve this goal, we have recommended relatively minor doctrinal adjustments designed to avoid this danger. If software is to be considered patentable, and clearly it is, these adjustments will help to ensure that the extension of patent protection achieves its intended effect.