Michigan Telecommunications and Technology Law Review

Volume 20 | Issue 2

2014

Structure from Nothing and Claims for Free: Using a Whole-System View of the Patent System to Improve Notice and Predictability for Software Patents

Holly K. Victorson
University of Michigan Law School

NOTE

STRUCTURE FROM NOTHING AND CLAIMS FOR FREE: USING A WHOLE-SYSTEM VIEW OF THE PATENT SYSTEM TO IMPROVE NOTICE AND PREDICTABILITY FOR SOFTWARE PATENTS

*Holly K. Victorson**

Cite as: Holly K. Victorson, Note, *Structure from Nothing and Claims for Free: Using a Whole-System View of the Patent System to Improve Notice and Predictability for Software Patents*,
20 MICH. TELECOMM. & TECH. L. REV. 497 (2014).
This manuscript may be accessed online at repository.law.umich.edu.

No uniform or customary method of disclosure for software patents is currently employed by inventors. This Note examines the issues that develop from software patent claims disclosed at various levels of abstraction, and the difficulties encountered by courts and the public when investigating the contours of the software patent space. While the courts have placed some restrictions on the manner in which software inventions are claimed, they are easily bypassed by clever patent applicants who desire to claim the maximum scope of their inventions. In the long run, however, a large "patent thicket" of overlapping and potentially overbroad inventions will work against the interests of inventors who desire to enforce their temporary technological monopolies. A confused field of inventions, when combined with a variety of abstracted disclosures, will result in unpredictable litigation and potentially invalidation of software as a patentable subject completely. Software patentees, like all patent holders, benefit greatly from robust and fair patent protection. In order to achieve the goal of a patent system that effectively balances the rights of the inventor with the right of the public in the interest of progress, however, this Note argues that patentees must recognize their critical role in the patent system and disclose the invention in code or pseudo-code at a level of abstraction that will allow the software community to effectively recreate the invention.

* J.D., University of Michigan Law School, 2015 (expected); B.S., Computer Engineering, University of New Mexico, 2006. The author would like to thank the Volume 20 editorial staff for their assistance in the editing process, especially Helen Ji, Carlyn Williams, Jason Wong, Daniel Zwick, Steven Beigelmacher, and Keith Lim.

INTRODUCTION

When the Framers of the Constitution drafted the "Intellectual Property" Clause,[1] they never could have anticipated the technological advances of the next two hundred years that would test the boundaries of that authorization. From "anticipatory package shipping"[2] to presenting recommended categories to a user based on her search query,[3] business methods utilizing advanced software techniques are improving the world around us.

Software is currently patentable in the United States. As a policy matter, however, its patentability is still subject to much debate.[4] Nevertheless, the innovative nature of software cannot be denied. From streaming Netflix movies from your phone to your thermostat detecting when you leave and automatically turning down the heat in the winter, software is responsible for many of the valuable developments in our lives. Because software lies at the heart of many of the advancements integral to our daily existence, novel developments should be protected by the patent system by granting a temporary monopoly over these technologies.

Unfortunately, because software is difficult to describe in traditional patent claims, courts have struggled with interpreting software claims and applying them to actual technologies in infringement actions. Disclosures by inventors vary wildly in their detail and exist at many levels of abstraction. As a result, "patent thickets" persist in certain technical areas, where many overlapping software patents exist in a certain field.[5] While the Federal Circuit has attempted to confine the breadth of software patents by enhancing the disclosure requirement for certain types of software claims, these requirements can be bypassed fairly easily by clever patent applicants who draft precise claims. This Note proposes taking a whole-system view of the patent system by acknowledging the critical importance of patentees to the fair and efficient operation of the patent system. Viewing software patents from such a perspective, the patent system should require a patent applicant to disclose relevant source code or pseudo-code in order to fully enable recreation of the novel portions of the invention. All four pillars of the patent system (the community of inventors and the executive, judicial, and legislative branches of government) should work together, utilizing the strengths of each group, to ensure this disclosure. Requiring such disclosure

---

1. U.S. CONST. art. I, § 8, cl. 8 (authorizing Congress "[t]o promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries").

2. *See* U.S. Patent No. 8,615,473 (filed Aug. 24, 2012).

3. *See* U.S. Patent No. 8,595,250 (filed Sep. 15, 2011).

4. See *infra* Section III.C., for a more detailed exploration of the anti-software patent movement.

5. *See* Gavin D. George, Note, *What is Hiding in the Bushes? Ebay's Effect on Holdout Behavior in Patent Thickets*, 13 MICH. TELECOMM. & TECH. L. REV. 557, 557 (2007).

will rebalance the careful bargain struck by the patent system of rewarding inventors who contribute to social progress by advancing technology.

In Part I, this Note walks through the background of patent law and the development of the current system of claiming an invention. Part II addresses how courts have interpreted functional software claims to date, and observes that while the Federal Circuit has placed some limits on means-plus-function claims, these limits are fairly easily thwarted by a clever claims drafter. Part III discusses why this claim system is insufficient for software patents and addresses some solutions that have been offered by others. Part IV proposes viewing the issue from a whole-system perspective and requiring disclosure of the source code at the novel points of the invention, which will help to restore order to the patent system and encourage further advancement of software innovations.

I. Background: Claiming, Functional Claiming, and Means-Plus-Function Claiming Under § 112(f)

Patent claims define the scope of the invention and determine the boundaries of the patent right enforceable by the patentee. The patent system has evolved to allow the inventor to claim her invention in a number of different ways. This Part explores how this evolution occurred, the limits that the courts and Congress historically placed on functional claims, and the options available to an inventor in describing her invention. Finally, this Part examines how the claiming options available to patentees may result in inconsistencies and confusion, particularly in software fields of invention.

Patent law attempts to achieve a "delicate balance . . . between inventors, who rely on the promise of the law to bring the invention forth, and the public, which should be encouraged to pursue innovations, creations, and new ideas beyond the inventor's exclusive rights."[6] Since the patent claims determine the boundary of the invention, broader claims grant a larger scope of protection to the patent owner. Since the early days of the patent system, courts have struggled to maintain the right balance between rewarding inventors for their contribution to the progress of science and protecting the public interest behind the Constitutional grant of power. Even though the prosecution of patents is an iterative process between the patent applicant and the examiner, the applicant has the greatest power to determine how an invention is disclosed and claimed. The scope of the patent protection is, therefore, largely determined by the decisions of the applicant herself.

Anyone who "invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof," can file a patent application with the United States Patent and Trademark Office (USPTO).[7] If the application meets all the legal

6. Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co., 535 U.S. 722, 731 (2002).
7. 35 U.S.C. § 101 (2012).

requirements, the applicant will receive a United States patent.[8] When an applicant submits her patent application to the USPTO, she must include, among other things, "one or more claims particularly pointing out and distinctly claiming the subject matter which the inventor . . . regards as the invention."[9] Claims define the invention and serve the dual purposes of communicating the patentability of the invention to the USPTO and drawing the boundaries of the invention for purposes of infringement determination.[10] An issued patent gives the applicant the right to exclude others from making, using, offering to sell, or selling the patented invention in the United States for a term of twenty years from the date of the original application.[11] A patentee can bring a civil action against any infringer and receive remedies including injunctions, damages, and, in exceptional cases, attorney fees.[12] The patent system thereby satisfies its Constitutional mandate of promoting innovation[13] by offering a legal remedy against unauthorized sales and uses of a recognized invention for a limited time.

The Supreme Court has deemed the temporary monopoly granted to a patentee "a property right," requiring clear boundaries in order to make clear to the public what exactly the patentee owns.[14] Clear boundaries allow the patent system to meet the Constitutional goal of promoting progress because it enables efficient investment in research and development.[15] Since the patent system grants limited monopolies to encourage innovation, clarity of the scope of the monopoly is important for both inventors and the public. If the boundaries of the limited property right are ill-defined, researchers may be hesitant to use the patent system to protect their invention, deciding instead to utilize trade secret protection and potentially depriving the public of the knowledge of the advancement. The public, too, should be able to rely on a valid patent document to gauge whether a particular article or good is infringing.

Originally, the patent system used a "central claiming" system to determine the scope of patent claims. In disputed cases, the court would examine the "essence" of the invention and determine what the heart of the invention was in order to construe the patent claims and determine whether infringement occurred.[16] In the nineteenth century, the patent system transitioned to

8. *Id.* § 111.

9. *Id.* § 112(b).

10. 3-8 DONALD S. CHISUM, CHISUM ON PATENTS § 8.01 (2013), available at LexisNexis.

11. 35 U.S.C. §§ 154(a)(2), 271(a).

12. *Id.* §§ 281, 283-85.

13. U.S. CONST. art. I, § 8, cl. 8 (giving Congress the authority "[t]o promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries").

14. *Festo*, 535 U.S. at 730-31.

15. *See id.*

16. *See* Michael Risch, *America's First Patents*, 64 FLA. L. REV. 1279, 1288 (2012).

a "peripheral claiming" system, in which the claims outline the outer borders, or periphery, of the invention.[17] This system was (and, generally, still is) believed to provide the public with better notice of the inventive boundaries captured by the patent, thereby encouraging more efficient investment in innovation.[18]

As the courts transitioned from central claiming to peripheral claiming, inventors increasingly began to use functional language to describe their inventions. Functional claiming allowed patent attorneys to draft claims to expand the boundaries of an invention to the fullest extent. For example, a claim for "a means for securing pieces of wood together" could allow an inventor to pursue infringers securing pieces of wood using nails, glue, screws, or any other adhesive means. Most inventors preferred to employ a functional claim over a claim covering a specific physical implementation to increase the value of the patent: if an inventor can claim ownership of the overall function of a machine, instead of a particular mechanical implementation that accomplishes the function, the inventor will be able to license the technology to a broader scope of users (or sue more potential infringers, as the case may be).

However, the patent system has always been careful not to allow overbroad functional claiming remove too much from the public domain. In order to reach a reasonable balance between rewarding inventors for their contributions and protecting the public from an unduly broad monopoly, courts limit the scope of overbroad patent claims that are unsupported by the accompanying disclosure. The Supreme Court, in the nineteenth-century case *O'Reilly v. Morse*, invalidated Morse's functional claim for transmitting information via electromagnetism when Morse did not "limit [himself] to the specific machinery, or parts of machinery, described in the foregoing specification and claims."[19] Morse's disclosure contained only one means of printing at a distance through electro-mechanical means, and the court noted that there may be other ways to accomplish the same function without using the specific device or method disclosed by Morse's patent.[20] Additionally, many circuits developed a "disclosure plus equivalents" restriction for functional claims, limiting the patentee to the structure (plus equivalents) for the functional claim disclosed in the specification.[21] This doctrine limited the patent owner's monopoly only to the specific embodiments of the invention

17. Dan L. Burk & Mark A. Lemley, *Fence Posts or Sign Posts? Rethinking Patent Claim Construction?*, 157 U. PA. L. REV. 1743, 1748-49 (2009).

18. *See* Jeanne C. Fromer, *Claiming Intellectual Property*, 76 U. CHI. L. REV. 719, 721 (2009).

19. O'Reilly v. Morse, 56 U.S. 62, 112 (1853).

20. *Id.* at 113.

21. *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, 240-42 (1999).

disclosed, along with any comparable variations of the invention easily ascertained from the disclosure.

The concern over the breadth of functional claims reached a peak when the Supreme Court attempted to ban all functional claiming, but Congress responded by allowing functional claims with limitations. In 1946, the Supreme Court eliminated functional claiming in *Halliburton Oil Well Cementing Co. v. Walker* by holding invalid claims that "do not describe the invention but use 'conveniently functional language at the exact point of novelty.' "[22] The Court expressed deep concern that clever patent attorneys and their clients were unfairly claiming more than they disclosed using broad functional claims, thus discouraging other inventors from experimenting in general technological areas and frustrating the purpose of the patent system.[23] In response, Congress restored limited functional claiming in the Patent Act of 1952,[24] which established the means-plus-function (or "combination") claiming style. This style allowed patentees to employ functional claims to capture an invention, but such claims would be construed to cover only the structure(s) disclosed in the patent specification and any equivalents. The statute, now codified in 35 U.S.C. § 112(f), states:

> Element in Claim for a Combination.— An element in a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure, material, or acts in support thereof, and such claim shall be construed to cover the corresponding structure, material, or acts described in the specification and equivalents thereof.[25]

Since a means-plus-function claim is interpreted in conjunction with the structure of the embodiments disclosed in the specification, it is critical that the structure disclosed in the patent is clear enough for a court to interpret in conjunction with the claimed function. Just as Morse could not claim to control all possible methods of transmitting information electromagnetically, a patentee awarded a means-plus-function claim is restricted to the particular structures disclosed in the specification.

All patent applicants have a great deal of control over how they claim their inventions, but inventors utilizing § 112(f) claims have additional control over how they present the corresponding structure to the function. Since

---

22. Halliburton Oil Well Cementing Co. v. Walker, 329 U.S. 1, 8 (1946) (quoting Gen. Elec. Co. v. Wabash Appliance Corp., 304 U.S. 364, 371 (1938)).

23. *Id.* at 11-12.

24. Although there are different theories, most scholars generally conclude that Congress specifically allowed for means-plus-function claiming in the Patent Act of 1952 to overrule *Halliburton*. Rudolph P. Hofmann, Jr. & Edward P. Heller, III, *The Rosetta Stone for the Doctrines of Means-Plus-Function Patent Claims*, 23 RUTGERS COMPUTER & TECH. L.J. 227, 243 (1997).

25. 35 U.S.C. § 112 (2012).

the inventor can control what does or does not go into her application, the quality of the patent largely depends on the diligence of the applicant. For example, imagine that a law professor creates a computer program that collects and aggregates information from the registrar regarding the dates and times of each of her courses and the names of the students signed up for each course. The program then randomly generates a set of students to cold call for each date of the course and produces a report. If our law professor filed a patent application on this invention in a means-plus-function style claim, it might look something like this:

> A law course management system, comprising:
>
> a means for receiving course information including course dates, course times, and students assigned to a particular course;
>
> a means for generating, at random, a set of students to cold call for each date of the course; and
>
> a means for generating a report containing the list of said randomly selected students.

In the specification for this application, the inventor would have to recite the "structure" in support of this claim in order to meet the requirement of § 112(f). If our inventor disclosed only a desktop-computer implementation in the specification and later sued another professor for taking her code and implementing the same method on a tablet, a court would likely limit the scope of her patent to the desktop structure disclosed unless the tablet was held to be an equivalent of a desktop computer. However, if the alleged infringer wrote a competing program for desktop computers that accomplished the same function but used the computer resources more efficiently, a court would likely hold the patent infringed. This result is inconsistent with the goals of the patent system: the blatant copier gets away, while the incremental improver is punished as an infringer.

II. The Courts and Functional Software Claims: On Unstable Ground

Part I introduced the concept of claiming and described how functional claiming evolved into the system that exists today. This Part looks at how the Federal Circuit treats software patents and why the formalistic treatment of claims may place the overall patentability of software at risk. Subsection A examines how the Federal Circuit has approached functional claims used in software patents and implemented a method of limiting the scope of claims written in means-plus-function format by requiring disclosure of an algorithm in the specification. Subsection A then examines how inventors and claims drafters can easily elude the original purpose of this enhanced requirement for software inventions. Subsection B looks at the Supreme

Court's potential rejection of software methods as patentable subject matter under § 101 as a result of the mess created by unclear software claims.

A. *The Federal Circuit Approach: Development of Enhanced but Ineffective Disclosure Requirements*

Today, functional claiming is commonly utilized by software patents.[26] Simply put, software is the set of instructions that runs on computer hardware to produce the images and text that are intelligible to the user. While the term "software" is commonly understood to mean code written in high-level programming languages like C++ or Java, it can also refer to the machine language instructions which tell the specific model of processor what low-level functions to perform or the assembly instructions that provide a more human-readable version of the machine language. There is no legal definition of a software patent. For the purposes of this Note, any patent for a function that is comprised primarily of algorithms designed to run on a processor is considered to fall under the umbrella of software patents. Since the patent system was designed to capture physical inventions, software patents fail to fit neatly into existing doctrine, and the Federal Circuit has struggled with software patent claims as a result. So far, the Federal Circuit has limited the scope of claims written in the means-plus-function format by requiring disclosure of an algorithm, but this limit is easily overcome by clever patent attorneys who can re-write the claims to fall outside of the scope of § 112(f).

Perhaps partially due to the negative publicity that some software patents have received, the Federal Circuit has recently restricted the broad reach of software patents that utilize means-plus-function claiming. In 1999, the Federal Circuit held, in *WMS Gaming, Inc. v. Int'l Game Tech.*, that the structure of software claims in means-plus-function format is the "special purpose computer programmed" with the software algorithm, not any "general purpose computer" that could be programmed to execute the patented function.[27] The Federal Circuit further asserted that a "general purpose computer" is transformed into a "special purpose machine" by the algorithm as implemented by the programmed code.[28] Accordingly, this opinion can be

26. Colleen V. Chien & Aashish R. Karkhanis, *Functional Claiming and Software Patents* 40 (Santa Clara Univ. Sch. of Law Legal Studies Research Papers Series, Paper No. 06-13, 2013), *available at* http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2215867). Chien & Aashish's study found that functional claims were more commonly found in software patents litigated by Patent Assertion Entities (PAEs), but about half of the litigated software patents involving non-PAEs also used functional claims.

27. WMS Gaming, Inc. v. Int'l Game Tech., 184 F.3d 1339, 1349 (Fed. Cir. 1999) ("In a means-plus-function claim in which the disclosed structure is a computer, or microprocessor, programmed to carry out an algorithm, the disclosed structure is not the general purpose computer, but rather the special purpose computer programmed to perform the disclosed algorithm.").

28. *See id.* at 1347-49.

interpreted to require the patent applicant to disclose an algorithm in order to adequately describe the special purpose machine for the purposes of § 112(f). This holding required software patentees who chose to use means-plus-function claims to disclose some semblance of an algorithm as part of the "structure" of the functional claim.

WMS then set the stage for *Aristocrat Technologies* in 2008, in which the Federal Circuit held that even if one of ordinary skill in the art might be able to develop a sufficient structure for a § 112(f) claim, the patentee must still "at least disclose the algorithm that transforms the general purpose microprocessor to 'a special purpose computer programmed to perform the disclosed algorithm.'"[29] In effect, even if a person of ordinary skill in the art could easily develop and implement an algorithm to meet the limitations of the claims, a patentee cannot escape the algorithm disclosure requirement for § 112(f) claims. Following the general premise that broad functional claims disclosing only a generic processor as structure do not limit the claim in any meaningful way, the Federal Circuit consistently held in later cases that a means-plus-function software claim limited only by the structure of a general purpose computer is invalid.[30] For example, consider how this system would apply to the cold call management system described in Part I above. Since she used functional claims to describe the invention, our inventive professor would likely now be required to disclose some sort of diagram, description, or code to describe the novel way in which students are randomly selected by the system.[31]

However, despite the enhanced disclosure requirement recently imposed by the Federal Circuit for software claims written in means-plus-function format, these requirements are largely toothless and easily avoided by a careful patent drafter. Patent applicants can still develop extremely broad functional claims that escape this enhanced scrutiny by re-writing the claims to avoid the court's formalist approach to categorizing § 112(f) claims. Furthermore, even if the claims are written in means-plus-function format, the patent applicant may choose to disclose an abstracted version of an algorithm[32] or, if the function is simple enough, disclose the abstract idea of a general purpose computer as the corresponding "structure" to the function.

29. Aristocrat Techs. Austl. Pty Ltd. v. Int'l Game Tech., 521 F.3d 1328, 1338 (Fed. Cir. 2008) (quoting WMS Gaming, 184 F.3d at 1349).

30. *See* Blackboard, Inc. v. Desire2Learn Inc., 574 F.3d 1371, 1382-83 (Fed. Cir. 2009); Finisar Corp. v. DirecTV Grp., 523 F.3d 1323, 1340-41 (Fed. Cir. 2008); Net MoneyIN, Inc. v. VeriSign, Inc., 545 F.3d 1359, 1365-67 (Fed. Cir. 2008).

31. For instance, if a student is generated by the system for a particular day, are they then removed from the pool of eligible students? May a professor select certain names to be chosen at longer average intervals (to account for the presence of students who voluntarily contribute to the class discussion)? There may be many novel ways to "randomly" select students from a set.

32. For example, through a prose-like description of the function or a generalized block diagram.

Patentees can manipulate claims to take advantage of the current system in at least three ways.

First, a patent applicant can take advantage of the formalist approach of the courts by redrafting the claims and avoiding means-plus-function scrutiny. The Federal Circuit limits software patent claims written in means-plus-function format, but software claims that are functional but do not conform to the standard § 112(f) format escape this scrutiny.[33] If a claim recites the term "a means for . . . ," the court reads in a rebuttable presumption that it should be interpreted under § 112(f).[34] If a claim does not use the "means for . . ." language, the claim is presumed to not be a means-plus-function claim. This creates uncertainty because the public is unable to predict how courts will evaluate software patent claims. Since patent applicants can structure their claims in any manner they choose, they may choose to use other language to avoid the automatic trigger of § 112(f) analysis in court. For example, consider our law professor from Part I, above. She might abandon the "a means for . . ." wording in her claim in favor of the following:

> A computerized method of managing a law course, comprising:
>
> receiving course information including course dates, course times, and students assigned to a particular course;
>
> generating, based on the course dates and student names, a set of random students to cold call for each date of the course; and
>
> generating a report containing the list of said random students.

By restructuring her claim in a manner that avoids using the words "a means for . . . ," she will escape a formalist application of a § 112(f) analysis and may even avoid the enhanced disclosure requirements imposed on the exact same claim written in means-plus-function format.

Second, even within the limits of § 112(f), a patent drafter has a significant amount of leeway to prepare potentially unclear claims. Even though the Federal Circuit requires disclosure of a step-by-step procedure as the "structure" of a claim under § 112(f), the disclosure can be accomplished in many ways, and disclosure of the software code is not required: a patentee can describe the function of the invention by describing it in prose or disclosing a flow chart or mathematical expression.[35] For a § 112(f) claim to be sufficiently definite, "a recited algorithm, or other type of structure for a

33. Mark A. Lemley, *Software Patents and the Return of Functional Claiming*, 2013 WIS. L. REV. 905, 920-28 (2013).

34. *See* TriMed, Inc. v. Stryker Corp., 514 F.3d 1256, 1259 (Fed. Cir. 2008) ("Use of the word 'means' in claim language creates a presumption that § 112 ¶ 6 applies.").

35. *Finisar*, 523 F.3d at 1340; *see also* Fonar Corp. v. Gen. Elec. Co., 107 F.3d 1543, 1549 (Fed. Cir. 1997) ("[F]low charts or source code listings are not a requirement for adequately disclosing the functions of software.").

[§] 112(f) claim limitation, need not be so particularized as to eliminate the need for any implementation choices by a skilled artisan; but it must be sufficiently defined to render the bounds of the claim—declared by [§] 112(f) to cover the particular structure and its equivalents—understandable by the implementer."[36] Because the algorithm required by the Federal Circuit can be disclosed in myriad of high-level ways, the requirement for algorithm disclosure from *Aristocrat Technologies* becomes inadequate for sufficiently defining the boundaries of a software patent. Since one skilled in the art of programming would be likely to have the ability to implement a high-level flow chart in a variety of ways, the public will have difficulty identifying the boundaries of these protected claims.

Finally, a patent applicant does not need to disclose an algorithm if a function can, by default, be performed by "any general purpose computer without special programming."[37] Aside from the question of what, exactly, constitutes a "general purpose computer," the fact that "processing," "receiving," and "storing" claims can be functionally claimed with only a "general purpose computer" disclosed as the structure for implementing the functions is troubling.[38] These claims, if valid, could be broadly construed by a fact-finder to expand the scope of the our professor's cold call processing system patent well beyond the system claimed, and fails to provide adequate notice to the public about the boundaries of the patent.

B. *The Supreme Court's § 101 Threat*

The trouble with software patents continues to frustrate courts and the court of public opinion. The longer the struggle goes on, the more likely the Supreme Court may simply resolve the confusion by holding software unpatentable subject matter under § 101.[39] This section of the Patent Act limits the type of invention available to be claimed by the inventor to "any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof."[40] The Supreme Court, in the 2010 *Bilski v. Kappos* case, had an opportunity to definitively state whether software and business methods were categorically patentable under § 101, but declined to do so.[41] Instead of stating a conclusive rule that software and business methods are patentable subject matter (or not) under § 101, the Su-

36. Ibormeith IP, LLC v. Mercedes-Benz USA, LLC, 732 F.3d 1376, 1379 (Fed. Cir. 2013).

37. *In re* Katz Interactive Call Processing Patent Litig., 639 F.3d 1303, 1316 (Fed. Cir. 2011).

38. *Id.*

39. *See* 35 U.S.C. § 101 (2012).

40. *Id.*

41. *See* Bilski v. Kappos, 130 S. Ct. 3218, 3227 (2010) (noting that until recently, patent law would have considered computer programs unpatentable subject matter under § 101, "[b]ut this fact does not mean that unforeseen innovations such as computer programs are always unpatentable").

preme Court held that a flexible application of the Federal Circuit's "machine-or-transformation" test should be applied to determine whether a claimed invention is a process under § 101.[42] This decision reversed an attempt by the Federal Circuit to limit patentable subject matter to only those processes that are "tied to a particular machine" or "transform[ ] an article into a different state or thing."[43] While the Supreme Court held the patent-at-issue—a method of hedging risk in commodities trading—invalid for being no more than an unpatentable "abstract idea," the majority made clear that business methods employing non-abstract methods may be patentable.[44]

Now, the Supreme Court has granted certiorari for *Alice Corp. Pty. Ltd. v. CLS Bank International* to consider whether certain software claims are abstract ideas ineligible for patent protection.[45] By doing so, the Supreme Court is again indicating interest in determining whether software methods are patentable subject matter. *Bilski* was a close call for software patents, but the subject matter narrowly escaped unpatentability. *Alice* could represent the menace on the horizon for software patents: there is a chance that this case may devastate their validity,[46] the overdue blowback resulting from a complicated subject area that the law simply has not yet figured out how to manage.

III. The Struggle with Software Inventions: Why There is No Easy Out

The unique characteristics of software naturally drive patent applicants to use functional claiming. As this Note explored in Parts I and II, those unique characteristics can be utilized by patentees to escape the court's attempt to appropriately limit the scope of software patents. This Part examines the abstraction characteristic of software, the issues that characteristic has caused in the patent system, and the proposed solutions to these issues. Subsection A delves into what exactly makes software abstract, and how that differs from other kinds of inventions. Subsection B considers why abstraction renders claim interpretation more difficult for courts. Finally, Subsections C and D consider possible resolutions to this tough problem.

---

42. *Id.*

43. *In re* Bilski, 545 F.3d 943, 961-62 (Fed. Cir. 2008), *aff'd*, 130 S. Ct. 3218 (2010).

44. *Bilski*, 130 S. Ct. at 3230, 3227.

45. Lyle Denniston, *Court to Rule on Patent Rights*, SCOTUSblog (Dec. 6, 2013, 12:35 PM), http://www.scotusblog.com/2013/12/court-to-rule-on-patent-rights.

46. The Electronic Frontier Foundation, in the amicus brief to the Supreme Court for Alice, encourages the Court to clearly set a standard for how section 101 should limit software patents. Brief of Amicus Curiae Electronic Frontier Foundation in Support of Grant of Petition at 1-3, Alice Corp. v. CLS Bank Int'l (2014)(No. 13-298), 2013 WL 5532728.

A. *The Abstraction Problem in Software Claiming*

Software patent applicants often use functional claims because the non-physical characteristics of the invention make functional claiming a natural choice. Unlike mechanical inventions, which are a means of implementing a function, software *is* the function. Software exists, and is defined, only by what it does. While the source code is comprised of characters that can be printed out and physically examined, this physical embodiment is just a representation of the software, not the software itself.[47] At the lowest level of abstraction, software is just a series of basic functions: compare the value at memory address `x` to the value at address `y`; add one to memory address `x`; compare the value at address `x + 1` to the value at address `y`. Even code written in high-level programming languages like Java or Ruby is eventually assembled and translated into machine code: instructions that a processor can understand and execute. Therefore, when a Java programmer codes an apparently simple high-level function like, in Java, `System.out.println(myString)`, the machine code that results from compiling this function is fairly complicated. This single instruction is translated into many lines of machine code to access the memory at the address where `myString` is located, read the characters of the string into registers, and call the platform-specific input/output subprogram to output the characters to the display device.

The functional aspect of software is often the novel and inventive part of the invention. Since the inventive function can be described in many different ways, software inventions are often inconsistently claimed by applicants, resulting in unclear boundaries of software patent grants. Because the USPTO does not require software patents to be written in any particular way (separate from the requirements applicable to all patents), the patentee is free to choose how to abstract her software invention in her patent claims and disclosure. For example, our inventive professor could disclose her software system by submitting source code, pseudo-code, function diagrams, system block diagrams, descriptive prose, mathematical formulas, or any combination thereof, in a § 112(f) means-plus-function style or a straightforward method style.[48] Of course, any patent application must pass the examiner's scrutiny of subject matter,[49] enablement,[50] novelty,[51] and non-obviousness,[52]

47. *See* RICHARD F. SCHMIDT, SOFTWARE ENGINEERING: ARCHITECTURE-DRIVEN SOFTWARE DEVELOPMENT 11 (Elsevier 2013) ("Software, as a substance for developing products, does not exhibit physical characteristics. Software is actually a language that is transformed into electrical currents within a processing unit that permits mathematical calculations.").

48. See *supra* Part II for a more detailed discussion.

49. 35 U.S.C. § 101 (2012).

50. *Id.* § 112.

51. *Id.* § 103.

52. *Id.*

but within those strictures, the patentee can abstract her invention disclosure in any manner she chooses.[53] The Manual of Patent Examining Procedure, a publication by the U.S. Patent and Trademark Office for patent examiners and attorneys, advises that examiners, when evaluating disclosure of "computer-implemented functional claims," should:

> . . . [D]etermine whether the specification discloses the computer and the algorithm (e.g., the necessary steps and/or flowcharts) that perform the claimed function in sufficient detail such that one of ordinary skill in the art can reasonably conclude that the inventor invented the claimed subject matter. Specifically, if one skilled in the art would know how to program the disclosed computer to perform the necessary steps described in the specification to achieve the claimed function and the inventor was in possession of that knowledge, the written description requirement would be satisfied.[54]

Black's Law Dictionary defines "abstraction," in relevant part, as follows:

> 1. The mental process of considering something without reference to a concrete instance <jurisprudence is largely the abstraction of many legal particulars>.
>
> 2. A theoretical idea not applied to any particular instance <utopia in any form is an abstraction>.[55]

Without more guidance, these definitions are rather unsatisfying when applied to software.[56] In a classic copyright case, Judge Learned Hand recognized that one could take any copyrighted work and make a series of copies, increasing in abstraction from the original work. Judge Hand further argued that, "there is a point in this series of abstractions where they are no longer protected [by copyright], since otherwise the playwright could prevent the

---

53. For example, compare U.S. Patent No. 6,727,830, col. 7-8 (filed July 12, 2002), claim 1 (Time based hardware button for application launch) with U.S. Patent 7,742,946, col. 11 (filed June 2, 2003), claim 1 (Advertising sales management system). Claim 1 of the '830 Patent claims "[a] method for expanding the functionality of an application button on a limited resource computing device," where claim 1 of the '946 Patent claims "[a] computer system for selling and allocating advertising time slots for a broadcast media" comprising a processor, memory, and an interface that performs certain structures. Both disclosures include logical flow charts to help describe the functional features, but the level of detail included on the flow charts differs (A decision point in Fig. 5 of the '830 Patent asks, "Have less than Z seconds elapsed since the identified application button timer was started?" A decision point in Fig. 7 of the '946 Patent asks, "Timeslot available?").

54. MPEP § 2161.01 (9th ed., Mar. 2014).

55. BLACK'S LAW DICTIONARY 10 (9th ed. 2009).

56. Indeed, since the word "instance" is used in object-oriented programming to describe a specific realization, or "instantiation," of a software object, it is somewhat confusing to refer to an idea without reference to an "instance" of something in this context.

use of his 'ideas,' to which, apart from their expression, his property is never extended."[57]

Similarly, software can be expressed in a variety of abstractions for the purpose of claiming a software invention. To take a picture using the camera software on my phone, I only need to point the sensor at something and press the correct button. One level down, the software receives a notification that it should take a picture, and then it calls existing function `takePhoto`. One more level down, the software registers my touch then calls the camera sensor to capture the light hitting it at that moment. When the sensor returns the data, the software makes another function call to send the data to memory. One final function is called to display the image I just captured on the screen. Even further down the abstraction chain, the ones and zeroes representing each pixel in the image are copied from a temporary storage area to the permanent memory on my device, and each memory location is checked to ensure the image is not overwriting a protected area of memory. Each of these steps is abstracted at different levels of the software to ensure that the next user up the chain is not bothered by the unnecessary details of the lower levels.

Software abstraction can broadly be categorized into two different flavors: process abstraction and data abstraction.[58] Process abstraction, well known and perhaps taken for granted by software developers, involves combining a series of processes into larger processing units in order to complete a commonly requested task. This prevents a programmer from repeating the same code, thus violating the "Don't Repeat Yourself" principle of programming,[59] and allows for higher levels of abstraction for the next user or developer (e.g., in C++, a programmer can call the `printf` function to send a string of characters to the standard output, which negates the need for the C++ programmer to understand how to send output to the specific display device).

Data abstraction can be accomplished by generalization, which is used in situations "in which concrete details in can be replaced by 'stand-ins' to create a reusable template."[60] This is often accomplished by creating a "container class" designed to keep pieces of data together and process them in a specific way.[61] For instance, our inventive law professor might make a generic `LawClass` class, with data containers to keep track of the time of

57. Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (3d Cir. 1930).

58. *See* Derek Bronish, Abstraction as the Key to Programming, with Issues for Software Verification in Functional Languages (2012) (unpublished Ph.D. dissertation, Ohio State University), *available at* https://etd.ohiolink.edu/ap/10?0::NO:10:P10_ETD_SUBID:76268.

59. Steve Smith, *Don't Repeat Yourself*, O'Reilly Commons, http://programmer.97things.oreilly.com/wiki/index.php/Don't_Repeat_Yourself (last visited Apr. 9, 2014).

60. Bronish, *supra* note 59, at 11.

61. *Id.* at 12.

the class, the location of the class, and the information for each student in the class. She might develop generic functions that would add a student to the class, remove a student from the class, give a student a grade, or generate a random set of names to cold-call for the day.

The other side of data abstraction is the reconceptualization of data generalization, where the programmer uses one or more generalized templates to perform the specific function that she desires. If our hypothetical law professor is teaching both torts and a product liability seminar this semester, she would create two new instantiations of `LawClass`, with data containing information specific to each class. She also might also modify the `whoToColdCall` function for the doctrinal torts class to ensure that every student is called at least once in the course before she calls on the same student for a second time.

Of course, an inventor in any technical field can abstract her invention in many ways. However, inventions grounded in a physical form still have a basic architecture that limits their functional breadth. The corresponding "architecture" of software inventions is the source code. The higher the level of abstraction used by the inventor to describe her software, the more the claims become purely functional, and the less clear the boundaries of the software invention become. This can implicate serious notice issues, as increasingly abstract inventions will be more difficult for the public to interpret. As a result, abstractly claimed inventions may impose a disincentive on technology developers in the area who will never know whether they may be subject to litigation for infringing some overbroad and unclear patent.[62]

B. *Claim Construction is Hard. Software Makes it Harder.*

Ultimately, courts may not be the best institution to interpret software claims. When the same claim can be written in a multiplicity of formats, district courts are more likely to diverge in their approach to evaluating similar claims. This generates uncertainty for both patentees and potential infringers, creating an unstable environment for research and development activities in software fields.

The variations of ways in which inventors can abstract their software patents increase the likelihood of litigation. Patent litigation, in general, is notoriously expensive and unpredictable.[63] Attorney's fees run into the many millions of dollars in cases where only one or two patents are in dis-

62. JAMES BESSEN & MICHAEL J. MEURER, PATENT FAILURE: HOW JUDGES, BUREAUCRATS, AND LAWYERS PUT INNOVATORS AT RISK 8-10 (2008).

63. *See, e.g.*, Stijepko Tokic, *Impact of Legal (Un)Certainty on Patent Valuation: What Investors Should Know Before Investing in Patents*, 22 FED. CIR. B.J. 363, 377 (2013) ("[I]nvesting in patents is a very risky business, not only because the monetary value of a patent is often uncertain, but also because patents can be invalidated, difficult to enforce, and subject to infringement lawsuits.").

pute,[64] and those fees only grow as the technologies and amounts in dispute increase.[65] Aside from the direct cost of legal fees, the time spent litigating is also expensive for any company involved in a patent infringement action. Due in part to the highly technical nature of the discovery involved and the tendency of the litigants to be highly contentious, a patent infringement case can take many months, if not years, to reach a decision at the trial court level.[66]

Even though patents are treated as property rights,[67] courts struggle to enforce those rights because the boundaries are often difficult to define.[68] The interpretation of these inventive boundaries is often a major area of dispute between litigating parties.[69] When the additional uncertainty stemming from the abstraction problem in software patents is added to this mix, a dangerous line-drawing problem emerges that may implicate the notice function of the patent system.[70] Additionally, as might be expected in a gray and rapidly changing area of the law, software patents have seemed to invite litigation. Indeed, the increased risk of litigation that comes with software patents has been well documented, and the astronomical amounts of money involved in software litigation has been a frequent topic in the news.[71]

---

64. Symposium, *A Panel Discussion: Claim Construction from the Perspective of a District Judge*, 54 CASE W. RES. L. REV. 671, 681 (2004) (District Court Judge Saris noted that her "run of the mill" cases cost $1.2 million in attorneys' fees to get through claim construction).

65. AIPLA, REPORT OF THE ECONOMIC SURVEY, 34 (2013) (Average litigation costs for a patent infringement action increased from $2.6 million, when the amount in dispute was $1-25 million, to $5.5 million, when over $25 million was in dispute.).

66. Jonathan L. Moore, *Particularizing Patent Pleading: Pleading Patent Infringement in a Post-*Twombly *World*, 18 TEX. INTELL. PROP. L.J. 451, 460-61 (2010).

67. *See* 35 U.S.C. § 261 (2012) ("Subject to the provisions of this title, patents shall have the attributes of personal property.").

68. Courts have long struggled with how to correctly interpret patent claims, even before software came into existence. *See* Topliff v. Topliff, 145 U.S. 156, 171 (1892) ("The specification and claims of a patent, particularly if the invention be at all complicated, constitute one of the most difficult legal instruments to draw with accuracy; and, in view of the fact that valuable inventions are often placed in the hands of inexperienced persons to prepare such specifications and claims, it is no matter of surprise that the latter frequently fail to describe with requisite certainty the exact invention of the patentee, and err either in claiming that which the patentee had not in fact invented, or in omitting some element which was a valuable or essential part of his actual invention.").

69. *See, e.g.*, ChriMar Sys., Inc. v. PowerDsine, Ltd., No. 01-74081, 2008 WL 2966470 (E.D. Mich. July 30, 2008) (construing thirteen terms in one order).

70. *See generally* Markman v. Westview Instruments, Inc., 52 F.3d 967, 997 (Fed. Cir. 1995), *aff'd*, 517 U.S. 370 (1996) ("[A] patent may be thought of as a form of deed which sets out the metes and bounds of the property the inventor owns for the term and puts the world on notice to avoid trespass or to enable one to purchase all or part of the property right it represents.").

71. *See* Colleen V. Chien, *Reforming Software Patents*, 50 HOUS. L. REV. 325, 332 (2012) ("[I]nternet patents are litigated about seven to ten times more often than average patents."); Joel Rosenblatt, *Apple Wins $290 Million From Samsung in Patent Retrial*, BLOOM-

Furthermore, since there is no common standard for writing software claims, courts may apply increasingly divergent standards to software patents during litigation. District Court judges see a wide variety of both criminal and civil cases. Even in non-software actions, generalist judges can feel overwhelmed by a technical area that they may not feel comfortable fairly evaluating. Judges have often expressed anxiety about adjudicating highly technical matters in patent cases.[72] Without a consistent expectation for what software claims should look like, courts are likely to apply widely divergent approaches to the interpretation of those claims, reducing predictability for litigants and increasing litigation costs.

Construing the meaning of the claims for software functions can be especially difficult. Even if two different algorithms complete the same functional steps, they are not necessarily equivalent. For instance, if one algorithm is written in assembly code and optimized for a particular processor's architecture, it will be significantly more resource-efficient than the exact same function written in assembly code and optimized for another processor's architecture. If you compare these two algorithms in machine code side-by-side, they will probably look significantly different.

Software claims are messy because of their functional nature.[73] However, software abstraction itself is not the problem. There will always be different levels of software abstraction, and that's a good thing: increasing levels of abstraction make it easier for programmers to create ever-more-complex software functions. However, for the purposes of disclosure of a patent, very high levels of abstraction (such as a requirements document) are never going to be adequate disclosure for software patents, because they only disclose the goals of the program instead of the manner in which those goals are accomplished.

C. *Software Inventions Are Not the Problem*

Citing a mismatch between the economic realities of the software industry and the patent system as it currently exists, some, including many

---

BERG (Nov. 21, 2013, 6:39 PM), http://www.bloomberg.com/news/2013-11-21/apple-wins-290-million-from-samsung-in-patent-retrial.html.

72. *See, e.g.*, Ass'n for Molecular Pathology v. U.S. Patent & Trademark Office, 702 F. Supp. 2d 181, 225 n.46 (S.D.N.Y. 2010) ("This author, confronted by genomics and molecular biology, also emphatically empathizes with Judge Hand's complaint in *Parke–Davis* about his lack of knowledge of the rudiments of chemistry."); Parke-Davis & Co. v. H.K. Mulford Co., 189 F. 95, 115 (C.C.S.D.N.Y. 1911) ("I cannot stop without calling attention to the extraordinary condition of the law which makes it possible for a man without any knowledge of even the rudiments of chemistry to pass upon such questions as these."). Even the typically self-assured Justice Scalia, in his concurrence in *Ass'n for Molecular Pathology v. Myriad Genetics, Inc.*, declined to sign on to the portion of the opinion detailing the finer points of molecular biology, because he was "unable to affirm those details on my own knowledge or even my own belief." 133 S. Ct. 2107, 2120 (2013).

73. *See supra* Section III.A.

software developers themselves,[74] support a radical change. Recognizing the costs involved with the U.S. patent system's struggle with software patents, these critics advocate completely eliminating software and business methods from the scope of allowable patentable subject matter.[75] However, this move would be a mistake. Software developments certainly can be profoundly innovative and worthy of patent protection. Rather, the issues identified by critics of software patents are indicative of problems with the patent system and its inadequacies in dealing with software inventions.[76] Eliminating software patents as a response to difficulties with (relatively) new technologies would throw the baby out with the bathwater. This move would make it more difficult for small businesses and individual inventors to find it economically feasible to innovate, creating barriers in a field that relies heavily on many small, incremental improvements for advancement.[77]

While many software developers, particularly in the Free and Open Source Software (FOSS) community, are against the principle of patenting software, the idea that some software should be available for anyone to use is not mutually exclusive with a properly functioning patent system. Since anything disclosed in an expired patent is considered part of the public domain, the patent system is one good way to distribute information about an innovation to the public and ensure continued open access. Even before the expiration of a patent, the owner could simply choose to dedicate the invention to the public or freely license the patent and decline to take action against potential infringers.[78] If FOSS developers engage in more defensive patenting and licensing instead of avoiding the system completely, innova-

74. The programming community, in particular, often perceives issued patents as overbroad, anti-competitive, and contrary to the promotion of innovation. *See, e.g.*, *When Patents Attack!*, THIS AM. LIFE (July 22, 2011), http://www.thisamericanlife.org/radio-archives/episode/441/when-patents-attack (noting that the software engineers interviewed loathed the patent system).

75. *See* Ben Klemens, *The Rise of the Information Processing Patent*, 14 B.U. J. SCI. & TECH. L. 1, 36 (2008); Wendy Seltzer, *Software Patents and/or Software Development*, 78 BROOK. L. REV. 929 (2013); Andrew Nieh, Note, *Software Wars: The Patent Menace*, 55 N.Y.L. SCH. L. REV. 295, 330 (2010).

76. *See* Mark A. Lemley & A. Douglas Melamed, *Missing the Forest for the Trolls*, 113 COLUM. L. REV. 2117, 2172-73 (2013) (noting that patent trolls are simply opportunists exploiting loopholes in the system, which is especially prevalent in the IT industry due to "broad functional claiming").

77. *See* BESSEN & MEURER, *supra* note 63, at 167-68 (describing the benefit of innovations from small inventors); Dan L. Burk & Mark A. Lemley, *Policy Levers in Patent Law*, 89 VA. L. REV. 1575, 1687-88 (2003) (describing software's relatively low cost, low risk, cumulative innovation process).

78. The Open Innovation Network is one example of an organization that is utilizing patents to further the goals of the F/OSS movement. *See* OPEN INVENTION NETWORK, http://www.openinventionnetwork.com/patents.php (last visited Apr. 9, 2014) ("Open Invention Network® acquires patents and makes them available royalty-free to any company, institution or individual that agrees not to assert its patents against the Linux System.").

tions in software will be more readily accessible to the public and will decrease the litigiousness of the area.[79]

While software certainly has its own unique quirks, and, to some extent, the patent system is still struggling with them, the underlying problem is not the innovative qualities of software. Like any other technology, software develops incrementally. Very few, if any, of these inventions are going to be "pioneering" like Edison's incandescent lamp or the Wrights' flying machine. Most inventions in any industry necessarily depend on those that came before—indeed, most software patents fall in line with the current state of the research in the field.[80] As long as software patents are disclosed in a disuniform manner, they will continue to be evaluated inconsistently by courts, thus creating uncertain boundaries and frustrating the delicate balance struck by the patent system between patentees and the public. These issues are not an indictment on the value or desirability of the patentability of software—instead, they are a sign that the patent system has not yet developed the right balance for software technologies.[81] If software patentees came to a common understanding of how to claim and describe their inventions, invention clarity would improve greatly, and courts would be able to better discern patent boundaries.

D. *Professor Lemley's Categorical § 112(f) Solution*

In a recent article, Professor Mark Lemley advocates that courts to "take seriously the dictate of [§] 112(f)" and analyze all software patents under the means-plus-function doctrine.[82] In order to be sufficiently definite under the current legal doctrine, a software patent must disclose the structure of "a computer programmed in a particular way," or the software itself, to meet the strictures of § 112(f).[83] "All [a court] needs to do is to take the statute at face value and limit functional claims to the particular way that the patentee implemented that function . . . with a particular computer program."[84] Others have similarly advocated for more stringent use of existing precedent.[85] Pro-

79. *See generally* Jason Schultz & Jennifer M. Urban, *Protecting Open Innovation: The Defensive Patent License as a New Approach to Patent Threats, Transaction Costs, and Tactical Disarmament*, 56 HARV. J.L. & TECH. 1 (2012) (arguing that open innovation communities must engage the patent system in a defensive manner in order to maintain or increase their presence in the innovation economy).

80. *See* Martin Campbell-Kelly & Patrick Valduriez, *A Technical Critique of Fifty Software Patents*, 9 MARQ. INTELL. PROP. L. REV. 249, 281 (2005).

81. *See id.* (finding that only six out of fifty representative software disclosures contained adequate disclosure).

82. Lemley, *supra* note 33, at 947-49.

83. *Id.* at 947.

84. *Id.* at 948.

85. *See* Elise S. Edlin, *Computer Claim Disarray: Untangling the Means-Plus-Function Doctrine to Eliminate Impermissible Functional Claiming in Software Patents*, 28 BERKELEY TECH. L.J. 417, 438 (2013).

fessor Lemley's proposal is attractive because it requires no action except a pronouncement from the Supreme Court or the Federal Circuit. The doctrine is already well established in the case law, so the court could require this new interpretation immediately, without any statutory modification.

While such an interpretation will help to provide a consistent evaluation of software claims in the court system, Professor Lemley's solution will not provide a meaningful limit on functional claiming in software patents. Courts will still struggle to construe software claims, and other structural issues will remain. In particular, the issues identified with the Federal Circuit's current application of the algorithm requirement for § 112(f) claims in Part II of this Note will continue to cause instability and unclear patent boundaries. Disclosure of an algorithm in prose or top-level block diagrams may not impose any meaningful limit or structure on the boundaries of the software invention. Professor Lemley appears to think that the Federal Circuit's current approach to the algorithm requirement for § 112(f) claims is doing a good enough job to appropriately limit the breadth of software claims,[86] but unless disclosure of the actual code or pseudo-code is required, the algorithm requirement is unlikely to have any true limiting effect. To his credit, Professor Lemley admits that the courts will still need to find the right level of abstraction for each invention.[87] However, the courts are just one pillar of the patent system, and an overreliance on just one of those pillars (and perhaps the slowest and most reactionary one) might result in further imbalances.

IV. A Whole-System View

Part III explored the challenges software patents pose within the current system and explained Professor Lemley's argument for an expanded application of the Federal Circuit's algorithm requirement for all software claims. While Professor Lemley's proposal is a good start by requiring algorithm disclosure as a part of any software application regardless of how the claims are written, it does not go far enough.

This Note argues that all actors within the patent system should work together to coordinate algorithm disclosure. In particular, the inventor should be recognized as a critical agent to help the entire system move more smoothly. This Note further proposes that an algorithm should be required as part of the disclosure for a software patent application only in the form of source code or pseudo-code at the point(s) of novelty. Since the appropriate level of abstraction may vary from invention to invention, the patent system must ultimately rely on inventors and patentees to disclose the algorithm at the most appropriate level of source code or pseudo-code.

86. Lemley, *supra* note 33, at 950.
87. *Id.* at 961.

A. *System-Level Duties: Leadership is Required from All Four Pillars of the Patent System*

While patent law policy analysts have seemingly endless suggestions for Congress, the courts, or the U.S. Patent and Trade Office to implement, the patentees themselves are often overlooked as the "fourth pillar" of the modern patent system.[88] Patent holders can be extremely influential in the continuing evolution of the patent system by the way they pursue patents, enforce valid patent rights, and license their inventions to others.[89] Instead of waiting on the political process to correct deficiencies in the system, patentees and patent applicants pursuing software patents can shape the system themselves by deciding to structure their future applications in a consistent manner. This idea is neither radical nor impractical: engineers and developers often team up to develop industry-wide standards to improve the quality of their products across the board.[90] Since they are the actors best placed to make the determination, software patentees should take responsibility for disclosing an algorithm at the correct level of abstraction. While the judicial system should impose a requirement of disclosure of an algorithm in conjunction with a software patent application, this requirement will likely be rendered meaningless unless patentees take collective responsibility for their part in the quality of issued patents.

The executive branch also has a role in the overall solution. The USPTO should work with patent applicants to encourage the disclosure of source code or pseudo-code at the point of novelty as the "structure" of software claims. This requirement would clearly define the scope of the patent and give a good starting point for evaluating equivalent implementations. Disclosing code will aid overloaded examiners in determining the patentability of a software application because they will have apples-to-apples comparisons to make. Examiners are trained in making the judgments necessary to determine patent eligibility, and the examiners assigned to software applications will be qualified enough to understand the disclosed source code and make a patentability determination.

Congress can certainly help, too: statutory clarification requiring code disclosure at the point of novelty will help remove the courts from the tangle of rules that they have developed over the years of struggling with functional software claiming. Congressional action will be immensely influential because the courts and, likely, the USPTO will avoid making any sweeping

88. Colleen V. Chien, *From Arms Race to Marketplace: The Complex Patent Ecosystem and its Implications for the Patent System*, 62 HASTINGS L.J. 297, 302 n.14 (2010).

89. *See id.* at 302-10 (describing the major ways that patent holders have historically influenced the development of U.S. patent policy).

90. The IEEE Standards Association even regulates technologies in its standards that are covered by patents. *See* UNDERSTANDING PATENT ISSUES DURING IEEE STANDARDS DEVELOPMENT, IEEE STANDARDS ASSOCIATION (2012), *available at* http://standards.ieee.org/faqs/patents.pdf.

changes in order to avoid surprising and disrupting the expectations of inventors, as harmful as those expectations may be in the long run.[91]

B. *Responsibilities of Patent Applicants: Disclose Code at the Appropriate Level of Abstraction*

This Note proposes that the patent system should require all software patent applicants to disclose the source code or pseudo-code at the point(s) of novelty. Part of the bargain struck by the patentee with the public, in exchange for a limited-term monopoly, is the complete and enabling disclosure of her invention. To meet her obligation to the public, the inventor must ensure that her invention is disclosed at an adequate level of abstraction. The correct level of abstraction required will depend on the invention. For example, an improvement to the efficiency of a compiler is going to be an invention at a different level of software abstraction than a new smartphone app. While flow charts and block diagrams are helpful to set the stage for describing a software invention, they do not tell the whole story. Even though a person who is skilled in the art may be able to take a flow chart and a block diagram and independently implement the function, without the source code (or pseudo-code), they are unlikely to be able to fully recreate the invention. The multiplicity of ways to implement even a simple function ensures that one recreating programmer may implement any given function more efficiently than another.

As part of the whole-system perspective advocated by this Note, the patent system must rely on applicants to disclose the code at the level of abstraction that makes the most sense to describe the novel aspects of the invention. At the point of novelty, the disclosure of the code will enable another software practitioner to implement and practice the entire invention. Accordingly, including code or pseudo-code for basic functions or functions already well-known in the art is not needed, unless the patentee is claiming a novel method of combining the existing art with the new development. It should be unnecessary, for example, to include millions of lines of code for a word processor if there are only one or two new features previously unknown in the art.

Disclosure of the source code or pseudo-code will go a long way to resolving the ambiguities that result from disclosure of the same algorithm in prose or block diagrams. The novel part of a valid software invention is a non-obvious improvement over what was previously known to the public. In order to ensure that the public is able to fully understand and reap the benefits of the improvement, the novel portions of the code must be disclosed. Accordingly, when the software community can more easily under-

91. *See Festo*, 535 U.S. at 739 (rejecting a major change to the interpretation of the prosecution history estoppel doctrine, in part, because it would "disrupt the settled expectations of the inventing community").

stand the invention, a court will be able to better determine the boundaries of the software patents, making the entire system more clear and predictable for software experts, courts, and businesses alike.

C. *Objections*

Some critics of requiring software to disclose code to adhere to a means-plus-function test note that requiring code could severely narrow software patents.[92] However, the risk of under-inclusion by requiring source code or pseudo-code is acceptable in the case of software inventions. While there is a chance that requiring software patents to include code in the disclosure will result in extremely narrow patents that will have reduced value for the inventor, this is an acceptable outcome if the result is still increased clarity in the field. The nature of innovation in the software field tends to be narrow, so it would make sense for the patent system to reflect that reality. The inventor herself is not the only concern of the patent system: the Constitutional goal is to "promote the progress of science and the useful arts," not to ensure all inventors are able to reap the maximum value from their patents. This goal is best served by providing more clarity for the boundaries of software patents.[93] Any unfairly narrowed claims resulting from the disclosure of code will be mitigated by the fact that the inventor of means-plus-function claims is statutorily entitled to protection of the exact structure disclosed and its structural equivalents in an infringement action.[94] Copyright also provides protection for expressions of ideas through a particular software implementation.[95] Furthermore, as the anti-software patent advocates like to note, patents are not required in the industry to encourage innovation. The FOSS movement has enjoyed a healthy existence for a long time. If the property rights to software are so inherently difficult to define that they can't be described in code, they are probably better off not being claimed as a property right that can be asserted against others for twenty years.

Further, some may criticize this approach exactly because disclosure of code can occur at multiple levels of abstraction,[96] creating an unpredictable expectation for the inventor and courts. However, this solution reflects the

92. *See, e.g.*, Robert E. Thomas, *Debugging Software Patents: Increasing Innovation and Reducing Uncertainty in the Judicial Reform of Software Patent Law*, 25 SANTA CLARA COMPUTER & HIGH TECH. L.J. 191, 235-36 (2009).

93. *See* BESSEN & MEURER, *supra* note 63, at 46-52.

94. 35 U.S.C. § 112(f) (2012). Even if software claims are not written in means-plus-function format, an infringer who makes only insubstantial changes to the patented invention will still be liable under the doctrine of equivalents. Charles Greiner & Co., Inc. v. Mari-Med Mfg., Inc., 962 F.2d 1031, 1035-36 (Fed. Cir. 1992).

95. *See, e.g.*, Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1253 (3d Cir. 1983).

96. *See* Kevin Emerson Collins, *Patent Law's Functionality Malfunction and the Problem of Overbroad, Functional Software Patents*, 90 WASH. U.L. REV. 1399, 1406 (2013)

nature of software innovation, which also occurs at multiple levels of abstraction. The inventor should disclose at the level that makes the most sense for the invention, and if the inventors are included and viewed as one of the pillars of the patent system, they are more likely to take their role in ensuring the viability of the system more seriously.

Conclusion

Software plays a crucial role in our everyday lives, and the incremental innovations that improve these technologies should be protected and incentivized. However, inventors must still live up to the bargain they agreed to when they applied for a patent by disclosing the source code for their invention at the points of novelty. This requirement will improve the predictability of intellectual property and encourage innovators in the field to research and experiment without fear of infringing a broad functional patent. As Professor Lemley notes, limiting the issues with software patents (or at least moving toward a solution) may end up saving software from complete invalidation under § 101.[97] If inventors and the other three pillars of the patent system can take responsibility for clarifying the boundaries of software patents, software innovations will continue to be protected by the system for many years to come.

(describing reliance on algorithms to limit the role in software as a potential "rabbit hole to infinite regress and madness").

97. Lemley, *supra* note 33, at 962-63.