1-1-2013

Fixing Software Patents

Eric Goldman
Santa Clara University School of Law, egoldman@gmail.com

Fixing Software Patents

By Eric Goldman[*]

January 2013

Abstract: This paper discusses software patents' unique attributes, challenges in trying to address problems with software patents, and some ideas for fixing the problems with software patents. It was written in connection with a November 2012 conference at Santa Clara University School of Law entitled "Solutions to the Software Patent Problem," and the initial version of this paper was published in three posts at the Tertium Quid blog at Forbes.

The U.S. patent system generally treats all innovations equally, but the innovation process varies widely across different industries. In particular, the software industry differs from other major innovative industries—such as computer hardware and biotech/pharmaceuticals—in several key ways, and those differences can create (and indeed have created) significant friction for the patent system.

Software patents have created big and expensive problems for companies throughout all sectors of our economy. Pretty much as soon as they get venture financing, start-up companies are approached by "patent trolls" with offers that can't be refused: pay off the troll now or pay a lawyer many times that amount later fighting in court. And large companies, especially in the smartphone industry, are paying literally billions of dollars to acquire patent portfolios to keep those portfolios from falling into the wrong hands; and with the hope that large patent portfolios will fend off competitor threats (*i.e.*, provide the company freedom to operate its business without interference from competitors' patents).

I wrote this essay in connection with the "Solutions to the Software Patent Problem" conference held at Santa Clara University (SCU) School of Law on November 16, 2012.[1] The conference was structured unusually for an academic conference. Rather than debate the merits of software patents, the conference started with the premise that software patents need to be fixed. This allowed us to spend all of our time exploring potential fixes, some of which are recapped in part three. To extend the discussion to a wider audience, many speakers also published short essays (designed to be accessible to lay readers) in Wired.com's Opinion feature.[2]

This essay proceeds in three parts. The first part explains how innovation in the software industry differs from other industries, and the implications of those differences for software patents. The second part explores two structural hurdles to addressing the unique problems that software innovations pose to the patent system: (a) the challenge of defining "software," and (b) which regulatory institution(s) can implement any fixes. The third part discusses some of my

[*] Professor of Law and Director of the High Tech Law Institute, Santa Clara University School of Law. Website: http://www.ericgoldman.org. Email: egoldman@gmail.com. This article combines three posts I made at my Tertium Quid blog at Forbes [http://blogs.forbes.com/ericgoldman] on November 28, December 11 and December 12, 2012. Thanks to Venkat Balasubramani, Colleen Chien, Kashmir Hill, Brian Love and Sruli Yellin for their comments.

[1] For more information, including links to the video recordings, presentation slides, articles, photos and more, go to the conference website, http://law.scu.edu/hightech/2012-solutions-to-the-software-patent-problem.cfm.

[2] *See* WIRED, http://www.wired.com/opinion/tag/solutions-to-the-patent-problem/.

favorite ideas proposed at the conference and how our community can make progress towards fixing software patents.

I. Why Are Software Patents Different?

What's unique about software? Some of the main differences that distinguish software from other innovations:

A. Software Has Short Innovation Cycles.

Software iterates quickly. Most software programs and their associated features have an effective commercial life of only a few years, at which point new software developments invariably render prior software innovations obsolete. Contrast this with mechanical innovations, which may have commercial lives of decades, and pharmaceutical innovations, which may retain commercial value indefinitely.

Two implications of the short innovation cycles in the software industry:

Software Has Significant First Mover Advantages. Software innovators can recoup some of their R&D investments from the de facto marketplace exclusivity associated with being the first mover. An example: assume that a particular software innovation has a two-year commercial lifecycle and it takes competitors 6 months to bring a matching product to market. In a situation like this, the first mover gets 1/4 of the maximum useful exclusivity period simply by being first to market. In some situations, the exclusivity period provided by the first mover advantage is more than enough to motivate software R&D without any patent protection.

Software Lifecycles End Before Patents Issue. As a practical matter, the commercial lifespan of many software programs and features is usually shorter than the time it takes the U.S. Patent & Trademark Office (PTO) to resolve a patent application—a process that can take four years or more. Invariably, the patented innovation will be obsolete by the time the Patent Office decides whether it's patent-worthy.

B. Software Will Be Developed Without Any Patent Incentive.

We principally justify patent law on utilitarian grounds: social welfare improves by providing innovators with an economic reward in the form of a limited-term marketplace exclusivity. The utilitarian argument leads to the hallmark "quid-pro-quo" of patents: society gives innovators valuable monopoly-like rights to exclude competition in exchange for a number of socially valuable deliverables like investments in R&D, public disclosure of those R&D results, and, eventually, the unrestricted rights to replicate the innovation.

When the quid-pro-quo model works as designed, it's great; but that's not always the case. One way the quid-pro-quo model can break down is when we give up the quid (*i.e.*, the rights to exclude) when we would have gotten the quo (*i.e.*, the R&D investments and public disclosure) in any case. In those circumstances, society "overpays" for the innovation.

Some reasons why we might be overpaying in the case of software patents:

Copyrights and Trade Secrets Provide Adequate Production Incentives. Multiple aspects of software can qualify for copyright protection, including source code, compiled code, visual layout, documentation, and potentially the aggregation of menu commands (the "structure, sequence and organization" of the software). The ideas embodied in those software attributes aren't copyrightable (they are protectable, if at all, under patent law), but the fuzzy borde3r between uncopyrightable ideas and copyrightable expression gives additional breadth to software's copyright protection.[3]

In addition to copyrights, trade secrets can protect software source code and potentially other aspects of software. While trade secret law doesn't prevent copying of widely available software, it can still slow the competition (and thereby extend any first mover advantage) by keeping certain aspects from helping competitors build their knockoffs.

Thus, even if it's not complete protection, the combination of copyright and trade secrets can provide substantial competitive protection for software. Historically, this combination has provided adequate incentives to the software industry. Indeed, during the first several decades of the software industry—a period which saw explosive industry growth—software patents were rarely obtained and even more rarely enforced.

Software Vendors Can Restrict Competition Without Patents. Software can have substantial lock-in effects that can hinder competition without patents. Software users may become locked into one vendor's offerings due to, for example, proprietary file formats, the difficulty of learning/re-learning menu commands or keystrokes, a developer community that creates valuable software-specific apps, and user investments in vendors' own proprietary customizations (such as the infamous example of user-authored macros in Lotus 1-2-3). We might lament the restrictions on competition caused by these lock-in effects, but so long as the software vendor doesn't break the law, they represent a crucial explanation for software innovation without patent incentives.

Software Gets Produced Without Any IP Incentives at All. The free and open source software community provides another example of software's quo without patent's quid. In those communities, contributors typically waive or voluntarily restrict any intellectual property protection for their contributions. Often, contributors monetize their efforts through non-IP mechanisms, such as offering maintenance or customization services for the software or building

[3] *See, e.g.*, Eric Goldman, *EA's Copyright Infringement Lawsuit Against Zynga Is Dangerous—For EA*, FORBES (Aug. 6, 2012 4:20 PM), http://www.forbes.com/sites/ericgoldman/2012/08/06/eas-copyright-infringement-lawsuit-against-zynga-is-dangerous-for-ea/, (discussing *Electronic Arts Inc. v. Zynga Inc.*, 3:12-cv-04099 (N.D. Cal. Aug. 3, 2012)), *and* Eric Goldman, *Recent Ruling in Triple Town/Yeti Town Game App Dispute Provides Cautionary Lessons for Both EA and Zynga*, FORBES (Sept. 27, 2012 11:53 AM), http://www.forbes.com/sites/ericgoldman/2012/09/27/recent-ruling-in-triple-townyeti-town-game-app-dispute-provides-cautionary-lessons-for-both-ea-and-zynga/ (discussing *Spry Fox LLC v. Lolapps, Inc.*, 2:12-cv-00147(W.D. Wash. Sept. 18, 2012)).

a reputation for programming expertise that leads to job offers.[4] Further, some free and open source software contributions are made purely altruistically, without any financial expectations at all. Collectively, the free and open source software community has proven that lots of software—even large-scale enterprise-class software—will be produced without any patent incentives at all.

C. Some Software Patent-Specific Problems

Software Gets Patented at Too High a Level of Abstraction. Stripped to its basics, software gathers, manipulates or displays data. There may be novel ways of accomplishing these goals, but the true novelty typically lies in how the software code is written, not the functional concepts of gathering, manipulating or displaying data that the code implements. Unfortunately, too many software patents claim protection at the highest level of abstraction (*e.g.*, "moving data on a network"), not at the more appropriate lower level like the more mundane implementation of that concept.

Software patent "over abstraction" leads to software patents that overclaim their true novelty. The over-claiming should lead to patent application denials on numerous grounds, including lack of enablement, failure to satisfy the written description requirement, and obviousness. But if the PTO doesn't aggressively screen software patents on those grounds, bogus patents get through the system. That's happened way too often.

The PTO Mishandled Software Patent Applications. Furthermore, for a long time—a decade or more—the PTO did not adequately research the prior art applicable to software patents. Patent examiners typically focus prior art searches on the database of existing patents. This means that when there's a watershed technological change—like the wave of software patent applications from the 1990s and early 2000s—the examiners don't see a lot of applicable prior art in their existing patent database. Therefore, the examiners grant patents that don't deserve protection. Eventually the patent database becomes rich enough with prior art to reach an equilibrium, but the errors in the interim produce a cohort of legacy patents that should never have issued.

Software Is Too Hard to Describe Precisely. Related to the abstraction problem, the boundaries of many software "innovations" are too hard to describe precisely. (In contrast, the specific implementation methods would be much easier to describe). Because of the semantic challenges, the resulting patents are so opaque that no one can understand what they mean. This also means that patent owners can adopt expansive interpretations of the patent boundaries and use the threat of patent litigation over these ambiguous and expansive borders to extract cash from potential defendants—even defendants who would ultimately prevail in court.

Patent Research by Subsequent Innovators Is Too Costly. In theory, product developers can check the patent database to see if they are transgressing someone else's patents. In practice, this doesn't work in the software industry for at least two reasons. First, as mentioned, the

[4] *See, e.g.,* Eric Goldman, *Wikipedia's Labor Squeeze and Its Consequences*, 8 J. TELECOMM. & HIGH TECH. L. 157 (2010), *available at* http://ssrn.com/abstract=1458162 (exploring alternative monetization methods by comparing Wikipedia with the FOSS community).

ambiguous wording of software patents means that subsequent developers may not interpret a patent to cover development efforts. Second, large software programs may have millions of lines of code, while patents may protect functionality that is expressed in only a few lines of code. As a result, a single software program could potentially implicate thousands, or even tens of thousands, of patents. The prohibitive costs to find these patents, research their applicability, and then, where appropriate, negotiate licenses to even a small fraction of those patents, would vastly exceed the potential economic returns from most software applications. Thus, software developers rationally choose not to research the patent database at all and instead "fly blind."

II. Challenges to Fixing Software Patents

The prior section described how software interacts with the patent system potentially differently than the way other innovative activities do. This section explores two structural challenges that will confront any effort to fix software patents: (A) defining the term "software," and (B) determining who can fix software patents.

A. Can "Software" Be Defined?

We may know software when we see it, but can we define "software" precisely enough to subject it to different treatment under patent law?

Consider the definition of "software patent" in a recently proposed bill, the SHIELD Act:[5]

> a patent that covers—
> '(A) any process that could be implemented in a computer regardless of whether a computer is specifically mentioned in the patent; or
> '(B) any computer system that is programmed to perform a process described in subparagraph (A).'

This definition isn't satisfying. First, the term "computer" doesn't have a natural boundary. Though elsewhere the statute defines "computer," the reality is that the definition may cover almost any electronic device—especially given that many devices increasingly have computer-like functionality and are networked. Second, I don't know what "process" means in this context, and the broad language "could be implemented" is uncomfortably capacious. *Just about anything* can be implemented via software.

In theory, we can distinguish software from physical devices (*e.g.*, "hardware"). But even if we do, innovators can often replicate software functionality by designing hardware to incorporate the functionality directly. In this sense, hardware and software are partial substitutes for each other. In fact, before patent law clearly allowed software patents, innovators (especially IBM) routinely obtained "software" patents by patenting hardware designed to perform the software-like function. So any special rules for software patents will just push innovators and their patent lawyers to seek patent protection for hardware that achieves the same outcome, obtaining the

[5] Saving High-Tech Innovators from Egregious Legal Disputes Act of 2012, H.R. 6245, *available at* http://www.govtrack.us/congress/bills/112/hr6245/text.

synthetic equivalent of a software patent. If that happens, we aren't making much progress on software patents.

Crafting special rules for software patents raises another concern. The U.S. has entered into international treaties that generally require us to treat all patentable innovations consistently, rather than providing heterogeneous protection for different innovative activity. Thus, trying to fix software patents might violate our international treaty obligations. Fortunately, Prof. Colleen Chien (SCU) has shown how we could implement software-specific fixes consistent with our treaty obligations.[6] Indeed, we've already created different rules for different innovations, such as special rules for business methods, surgical methods, tax strategies and others.

So, fixing software patents is tricky. It may not be possible to define software patents precisely, it may be easy for patent applicants to game any software-specific rules, and we have to find a way to remain in compliance with our treaty obligations. On the other hand, if we avoid software patent-specific fixes and instead try to make changes across all patents, that would dramatically increase the number of potential opponents to the change—and reduce the odds of success.

B. Who Can Fix the Problems?

Whether we decide to change software patents specifically or all patents categorically, the other major threshold issue is figuring out *who* can implement a fix. The main options are Congress, the courts, the PTO and the industry itself through market solutions and industry self-regulation.

Congress. In general, the most elegant solutions to software patents involve changes to the patent statute, although most advocates are reluctant to explore this route. After all, it took years to get the America Invents Act (AIA) through Congress, and the result was significantly watered down from its starting point (and unusually buggy).[7] Congress probably doesn't have a lot of enthusiasm right now to tackle big structural revisions to patent law, and there's always the possibility of further legislative paralysis based on pushback from the pharmaceutical industry, trolls or the few software industry players who are OK with software patents.

As an intermediate compromise, at the SCU conference Prof. Christal Sheppard (Nebraska) suggested that Congress could pass more resolutions stating Congress' intent. Although the resolutions aren't binding law, they nevertheless could provide helpful guidance to the courts more quickly and without the typical bruising fights.

The Courts. Law professors often prefer using the judicial system to effectuate legal change. Common law courts have the inherent power to evolve the law, and judicial evolution sidesteps the messy legislative sausage-making process. But courts generally can only color within the lines laid for them by Congress (subject to possible Constitutional limits on legislative power), so courts can't solve all of the problems themselves. Judges also attempt to resolve the disputes

[6] *See* Colleen V. Chien, *Tailoring the Patent System to Work for Software and Technology Patents* (Nov. 15, 2012), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2176520.
[7] *See* An Act to Correct and Improve Certain Provisions of the Leahy-Smith America Invents Act and Title 35 United States Code, H.R. 6621, *available at* http://www.govtrack.us/congress/bills/112/hr6621.

on their docket, not broader social issues, and they often don't (and shouldn't) seek out information not provided by the litigants fighting those disputes. Thus, it's not especially easy to educate judges and get them to reflect broader social concerns in their rulings.

PTO. As the government agency that manufactures patents, the PTO has a lot of power to control its manufacturing process. However, the PTO generally leans pro-patents and therefore may lack enthusiasm to fix any problems that result in over-patenting. The PTO also must abide by the rules established by Congress and the courts.

Industry Self-Regulation. For many software industry participants, software patents create a type of prisoner's dilemma.[8] If the industry could find a way to avoid software patents, the overall industry would be better off. However, so long as any company can obtain patents, it's in that company's best interest to do so (whether for offensive or defensive purposes). The result is that each company in the industry acquires patents solely to maximize its individual welfare, but the entire industry is worse off. A successful market-based approach would change companies' payoffs so they no longer face the prisoner's dilemma. This is easier said than done.

In all likelihood, we may need participation from each institution to fully redress the problems with software patents.

III. Ways to Fix Software Patents.

Unfortunately, the rhetoric about software patents tends to devolve into a black-or-white debate: Are software patents good for society, or should we kill them?

We need to move past that binary—and irresolute—discussion. At the SCU conference, we simply assumed that software patents were a problem. With that premise, conference participants could focus their attention on thoughtful and creative ways to redress the problems created by software patents. Undoubtedly, some of the speakers and attendees favored killing software patents outright. However, most of the speakers proponed more nuanced and implementable approaches to remediate the problems of software patents, without resolving the question of whether software patents are good or bad.

At the SCU conference, we heard *a lot* of different proposals (nearly two dozen) from experts in the field. All of them had promise (that's why we put the presentations on the agenda in the first place), but here's a recap of some of my personal favorites:

Prof. Mark Lemley (Stanford). Prof. Lemley argued that many software patents use "functional claiming," which is patenting a software function (the problem that the patent seeks solve) rather than a specific way to implement that functionality (the innovator's solution to the problem). For example, currently we allow patent claims in the form, "a computer programmed

[8] *See Prisoner's Dilemma,* WIKIPEDIA, http://en.wikipedia.org/wiki/Prisoner%27s_dilemma (last visited Dec. 21, 2012) ("The prisoner's dilemma is a canonical example of a game analyzed in game theory that shows why two individuals might not cooperate, even if it appears that it is in their best interests to do so."). *See also* Avinash Dixit and Barry Nalebuff, *Prisoner's Dilemma*, LIBRARY OF ECONOMICS AND LIBERTY, *available at* http://www.econlib.org/library/Enc/PrisonersDilemma.html.

to achieve this result" or "a computer programmable/capable of achieve a result" (his research identified 11,000 patents using the "capable of" language). Prof. Lemley argued that we should prevent functional claiming in software, allowing patents only on methods of achieving the function, not the function itself.[9]

Prof. Lemley's proposal squarely attacks the breadth of software patents. By limiting software patents to their specific way of accomplishing a function, other innovators can develop alternative solutions without infringing the patent. Further, courts and the PTO can execute this solution without legislative changes by changing the way they apply existing laws. Thus, this solution could be implemented immediately and cheaply.

Prof. Arti Rai (Duke). Prof. Rai noted parallels between software patents and patents on bioinformatics—basically, software applied to biopharmaceuticals—which are handled in PTO Art Unit 1631.[10] In art unit 1631, the PTO implemented a more stringent non-obviousness review of applications, requiring patent applicants to provide more thorough written patent descriptions and more definite patent claims. Due to the heightened scrutiny of bioinformatics patents, a noticeably higher percentage of applications were rejected on non-obviousness grounds. Prof. Rai argued that software could be subjected to similarly heightened scrutiny, something that (like Prof. Lemley's solution) the PTO and courts could implement immediately under existing law. On the other hand, changing the PTO's evaluation of software patents doesn't address the perniciousness of legacy software patents that never should have issued.

Prof. John Duffy (Virginia). Prof. Duffy took a more conceptual approach than Profs. Lemley and Rai. He argued that patent law's non-obviousness requirement should screen out innovations that multiple inventors identify around the same time.[11] While perhaps counter-intuitive, simultaneous invention happens often.[12] If multiple inventors achieve the same solution around the same time, then the innovation apparently was obvious to the relevant experts and therefore doesn't deserve the patent reward.

Though the approach isn't limited to software patents, it would help with software patents—which are also the subject of simultaneous invention—and it could be implemented by the PTO and judges without new legislation. However, I'm concerned about the adjudication costs to

[9] *See* Mark A. Lemley, *Let's Go Back to Patenting the 'Solution,' Not the 'Problem'*, WIRED (Oct. 31, 2012 6:30 AM), http://www.wired.com/opinion/2012/10/mark-lemley-functional-claiming/ *and* Mark A. Lemley, *Software Patents and the Return of Functional Claiming* (Stanford Public Law Working Paper No. 2117302, July 25, 2012), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2117302.
[10] *See* Arti K. Rai, *Let's Tame Software Patent Claims: Lessons from Bioinformatics*, WIRED (Nov. 20, 2012), http://www.wired.com/opinion/2012/11/software-patents-bioinformatics/.
[11] *See* Michael Abramowicz & John F. Duffy, *The Inducement Standard of Patentability*, 120 Yale L. J. 1590 (2011), http://www.yalelawjournal.org/images/pdfs/974.pdf, *and* John F. Duffy, *Let's Get Rid of Kludgy Patent Fixes and Define the Non-Obvious*, WIRED (NOV. 16, 2012), http://www.wired.com/opinion/2012/11/lets-get-rid-of-kludgy-patent-fixes-and-define-the-non-obvious/.
[12] *See* Malcolm Gladwell, *Who Says Big Ideas Are Rare?*, THE NEW YORKER (May 12, 2008), *available at* http://www.newyorker.com/reporting/2008/05/12/080512fa_fact_gladwell/?currentPage=all, Mark A. Lemley, *The Myth of the Sole Inventor* (Stanford Public Law Working Paper No. 1856610, Jul. 21, 2011), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1856610, *and Multiple Discovery*, WIKIPEDIA, https://en.wikipedia.org/wiki/Multiple_discovery (last visited Dec. 21, 2012).

figure out if multiple inventors made the same innovation around the same time. Furthermore, I would expect patent applicants in some non-software disciplines—notably, pharmaceuticals—may resist this approach, potentially even seeking Congressional intervention.

Prof. Sam Vermont (University of Miami). Prof. Vermont noted that in some cases, the research costs to find and review precedent patents are more expensive than independent invention. This is especially true with software innovations.[13] Thus, Prof. Vermont proposed a defense to infringement in cases where "defendant's costs to find the patentee's version of the invention beforehand were greater than the defendant's costs to invent it on his own."[14] In some sense, this situation is a damning indictment of those patents; if it's cheaper to recreate the innovations than to find their patents, why did we give the patent reward to those innovations in the first place? For that reason, I like Prof. Vermont's suggestion. However, it would require new legislation to implement, making it potentially more of an insightful thought exercise than a practical near-term solution.

Profs. James Bessen (Boston University) and Brian Love (SCU). Professors Bessen and Love separately advocated for maintenance fee reform as a solution to software patents. Prof. Bessen analogized patents to pollution because both produce negative externalities. To internalize the externality, Prof. Bessen argued for Pigouvian taxes[15] in the form of increased maintenance fees (*i.e.*, ongoing payments to maintain the patent in effect). By his estimate, the maintenance fee rates should increase tenfold to reflect the patents' true social costs. He would further adjust maintenance fees to reflect the likelihood of patents being asserted, which would lead to even greater increases for software and business method patents.[16]

In a separate project, Prof. Brian Love has identified that some of the lowest-merit patent lawsuits are brought near the end of a patent's term. We might avoid some of those suits by statutorily truncating the patent term, but Congress isn't likely to approve that. Instead, we can achieve a similar result by changing the maintenance fee structure to financially encourage some weak patents to lapse early, increasing maintenance fees as the patent ages, and adding new late-term maintenance fee payment periods.[17]

---

[13] *See, e.g.*, Brad Burnham, *We Need an Independent Invention Defense to Minimize the Damage of Aggressive Patent Trolls*, UNION SQUARE VENTURES (Jan. 11, 2010), http://www.usv.com/2010/01/we-need-an-independent-invention-defense-to-minimize-the-damage-of-aggressive-patent-trolls.php (discussing the high frequency of simultaneous invention in the software industry).

[14] Samson Vermont, *No Social Harm, No Legal Foul*, https://dl.dropbox.com/u/30084723/No%20Social%20Harm%2C%20No%20Legal%20Foul.docx (last visited Dec. 21, 2012).

[15] *See, e.g.*, *Pigovian Tax*, WIKIPEDIA, https://en.wikipedia.org/wiki/Pigovian_tax (last visited Dec. 21, 2012) ("A Pigovian Tax (also spelled Pigouvian Tax) is a tax applied to a market activity that generates negative externalities").

[16]*See* James Bessen, *Can New Fees Fix the Patent System?: Experts Weigh In*, WIRED (Sept. 6, 2012 2:10 PM), http://www.wired.com/opinion/2012/09/can-new-fees-fix-the-patent-system/.

[17] *See* Brian Love, *An Empirical Study of Patent Litigation Timing: Could Patent Term Reduction Decimate Trolls Without Harming Innovators?*, 161 U. PA. L. REV. __ (forthcoming 2013), *available at* http://ssrn.com/abstract=1917709 *and* Brian Love, *How PTO Fees Could Decimate Patent Trolls* (Working Paper, Nov. 14, 2012), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2175870.

Economic theory supports modifying maintenance fees by getting patent owners to internalize the costs of their actions. In particular, I was particularly persuaded by Prof. Love's analysis that we should have at least one more maintenance fee payment period later in the patent term; the last maintenance fee payment date comes too early in the patent term.

However, tinkering with maintenance fees could produce unexpected results. A *Vanderbilt Law Review* paper[18] suggests the PTO may adjust its patent issuance rate to maximize its revenue. Reducing the PTO's maintenance fee revenue by lapsing more patents might cause the PTO to issue more patents to increase revenues from patents before they lapse. Similarly, increasing patent maintenance fees could cut in different directions. The PTO might reject weak patents because it's receiving increased revenues from issued patents, or the PTO might issue more weak patents in a pure revenue grab. It might be possible to ameliorate these unwanted budget-driven decisions, but before we tinker with maintenance fees, we need to understand all of the economic effects, not just how patent owners will respond.

Prof. Jennifer Urban (UC Berkeley). Prof. Urban, along with Prof. Jason Schultz (UC Berkeley), proposed the Defensive Patent License (DPL).[19] Patent owners that opt-into the DPL agree to license their patents royalty-free to everyone else who has opted into the DPL. In effect, everyone participating in the DPL agrees not to sue each other for patent infringement. This gives companies substantial incentive to opt-in to the DPL; by doing so, they eliminate the potential patent risk from dozens, hundreds or even thousands of other industry participants. From a theoretical perspective, this solution nicely responds to the industry's prisoner's dilemma by providing incentives to voluntarily defang patents.

However, the DPL has some obvious limitations, including: (1) it does not affect patent trolls who would likely never participate in the DPL, (2) it assumes enough players join the DPL to make it attractive for other companies to want to participate (*i.e.*, the DPL exhibits network effects), and (3) it assumes that DPL participants bring enforcement actions against non-DPL participants; otherwise, non-participants can free-ride off the DPL network without paying the costs. I think the DPL is a brilliant concept, but it remains to be seen how effective it will be in the marketplace.

At the SCU conference, Ben Lee of Twitter introduced a related idea, the Innovator's Patent Agreement (IPA),[20] where Twitter unilaterally promises that it will not assert specific patents offensively. This promise principally is aimed at Twitter's own employees, who don't have to fear that their innovative work will lead to socially harmful patent lawsuits. The IPA doesn't rely on network effects, so any company can adopt it unilaterally. For that reason, however, IPA adoptions may be slow and possibly *de minimis*.

[18] *See* Michael Frakes & Melissa F. Wasserman, *Does Agency Funding Affect Decisionmaking?: An Empirical Assessment of the PTO's Granting Patterns*, 66 VAND. L. REV. __ (forthcoming 2013), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1986542.
[19] *See* Jason Schultz & Jennifer M. Urban, *Protecting Open Innovation: A New Approach to Patent Threats, Transaction Costs, and Tactical Disarmament*, 26 HARV. J. L. & TECH. __ (forthcoming 2012), *available at* https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2040945.
[20] *See* Adam Messinger, *Introducing the Innovator's Patent Agreement*, TWITTER BLOG (Apr. 17, 2012), http://blog.twitter.com/2012/04/introducing-innovators-patent-agreement.html.

Other Ideas. Two other solutions to software patents that came up throughout the SCU conference include the SHIELD Act and an independent invention defense.

The SHIELD Act. The SHIELD Act[21] would make it easier for successful defendants to recoup attorneys' fees in weak computer patent cases (both hardware and software). The fee-shift only benefits defendants; successful plaintiffs wouldn't be any more likely to get their attorneys' fees covered. As a result, the SHIELD Act would increase plaintiffs' anticipated costs of litigation and thus discourage weak computer patent suits.

None of the speakers at the SCU conference expressly advocated for the SHIELD Act, although Google's Kent Walker (and others) did express support.[22] Like all legislative solutions, the odds that the SHIELD Act will pass in the near future are low.

An Independent Invention Defense. Independent invention, *i.e.*, innovators developing the same innovation without reference to each other's work, occurs all the time, especially in the software community. Unlike the other main intellectual properties,[23] patent law does not have a general purpose independent invention defense. Why do we treat patents differently?

An independent invention defense to patent infringement poses several practical problems. First, it will be asserted by most, if not all, defendants, thereby increasing patent adjudication costs. Second, the evidence to prove or disprove independent invention often will be self-serving and not credible. If we ask engineers to self-report what material they reviewed to achieve their goals, invariably they won't admit looking at the plaintiffs' material. Third, the defense will give inventors even more incentive not to research the patent database for fear of seeing patents and thus losing the ability to claim independent invention. That would further undermine the public disclosure benefits of patents.

These problems highlight the strength of Prof. Duffy's suggestion. Rather than trying to investigate what was independently invented and what wasn't, we could use the fact that multiple inventors reached the same solution to support a non-obviousness objection—and, unlike an independent invention defense, we could achieve that result without any legislative changes.

---

[21] Saving High-Tech Innovators from Egregious Legal Disputes Act of 2012, H.R. 6245, *available at* http://www.govtrack.us/congress/bills/112/hr6245/text.

[22] *See* Kent Walker, *Google: Don't Let Trolls Exploit Patent System Flaws*, WIRED (Nov. 19, 2012, 3:30 PM), http://www.wired.com/opinion/2012/11/google/.

[23] Independent invention is a complete defense to trade secret misappropriation, although a person actually exposed to trade secrets is effectively barred from later claiming independent invention of those ideas. Independent creation is also a defense to copyright infringement; but like trade secrets, a person exposed to a copyrighted work can't claim independent creation, and furthermore, we inferentially presume copying rather than independent creation in some situations where the defendant may have been exposed to the work. Trademarks aren't infringed merely by copying, so independent development of an identical trademark does not contribute to trademark infringement However, intent to copy a third party trademark is a factor that weighs against defendants; proving good intent goes a long way towards a successful defense. *See* Barton Beebe, *An Empirical Study of the Multifactor Tests for Trademark Infringement*, 94 CAL. L. REV. 1581 (2006), *available at* http://scholarship.law.berkeley.edu/californialawreview/vol94/iss6/1.

What's Next?

Our November SCU conference attracted over 250 attendees—most of whom want to fix the problems with software patents. It was a remarkably large and enthusiastic crowd, and there was palpable energy to fix software patents. It's clear that the Silicon Valley community, and many other technology communities, is experiencing a lot of pain from software patents.

Even so, I remain bearish on legislative solutions—even relatively minor changes like the SHIELD Act—at least in the next few years. Congress made some big moves with the AIA, and just cleaning up the AIA's remaining bugs will consume whatever remaining time and appetite Congress has for patent reform.

As a result, I'm more optimistic about solutions that the PTO or judges can unilaterally and immediately implement without any statutory changes. I think a combination of Profs. Lemley's, Rai's and Duffy's proposals are especially promising. By re-characterizing the appropriate level of acceptable abstractness, requiring more proof from patent applicants, and then screening out innovations that multiple inventors achieve around the same time, the PTO would kill most of the least appropriate patents. Judges could then finish off any unmeritorious patents that somehow get through the PTO. I also think marketplace solutions, like the DPL and IPA, hold promise, but aren't likely to solve the problems on their own.

Ultimately, the single most important thing we can do is to keep identifying, publicizing and discussing problems with software patents, such as Prof. Chien's research on startups and patent trolls[24] and reforming software patents.[25] By continuing to shine the spotlight on the issues, policymakers and judges won't be able to ignore them—as evidenced by a recent speech by PTO Director Kappos shortly after the SCU conference,[26] where he simultaneously (and quite defensively) decried critics of software patents and outlined the many steps the PTO has taken to improve handling of software patents. Perhaps it's so obvious that it need not be said at all, but the many criticisms levied against the PTO's handling of software patents have helped drive some of the PTO's improvements. Simply discussing software patent issues publicly makes a difference.

[24] *See* Colleen V. Chien, *Startups and Patent Trolls*, Santa Clara Univ. Legal Studies Research Paper No. 09-12, Sept. 28, 2012, *available at* http://ssrn.com/abstract=2146251.
[25] *See* Colleen V. Chien, *Reforming Software Patents*, 50 HOUSTON L. REV. __ (forthcoming 2012), *available at* http://ssrn.com/abstract=2125515.
[26] *See* David Kappos, *An Examination of Software Patents*, Keynote Address at the Center for American Progress (Nov. 20, 2012), *available at* http://www.uspto.gov/news/speeches/2012/kappos_CAP.jsp.