A New Look at Patent Reform: Comparison to other suggested approaches

Lee A. Hollaar
Professor, School of Computing
University of Utah

March 9, 2007 version

Introduction

Since I first wrote the paper, two particularly noteworthy suggestions have been published. But neither really addresses the problems I noted above, and one would likely cause problems with copyright while not substantially helping patents.

"Gold-plated patents"

In their paper "What to Do about Bad Patents?,"[1] Profs. Mark Lemley, Doug Lichtman, and Bhaven Sampat suggest a tiered patent system. Their proposal recognizes that very few patents are actually litigated, or even asserted against another party.[2] There is little point in improving the examination for every patent if there could be some way of identifying important patents, and they propose to do that by seeing if somebody has enough interest to pay for a better examination.

That would come about in one of two ways. Either the patent owner may pay a substantially-larger fee for a more comprehensive examination, or another party may require a reexamination of the patent. As the authors write, "applicants should be able to 'gold-plate' their patents by paying for the kind of searching review that would merit a presumption of validity."[3]

---

[1] *Regulation*, Vol. 28, No.4, Winter 2005-2006, pp 10-13, available at http://www.cato.org/pubs/regulation/regv28n4/v28n4-noted.pdf.

[2] They point to recent bad patents: "obvious inventions like crustless peanut butter and jelly sandwiches, ridiculous ideas like a method of exercising a cat with a laser pointer, and impossible concepts like traveling faster than the speed of light."

[3] There are two aspects to the statutory presumption of patent validity. The first is the "burden of persuasion" which requires that the challenger of a patent must put forth evidence of invalidity, rather than the patent owner having to prove validity as part of its infringement claim. This only makes sense, because it is impossible for a patent owner to show that there is no prior art anywhere in the world that would render the patent invalid. This concept is illustrated by a court finding a patent "not invalid," rather than "valid," if the defendant is unsuccessful in proving invalidity.

The second aspect is the "burden of producing evidence," which establishes the sufficiency of the evidence required to prove invalidity. Patents now enjoy the heightened requirement of clear and convincing proof, more than the normal civil litigation standard of the preponderance of the evidence.

The proposal does nothing to reduce the long pendency without protection that is a special problem for fast-moving technologies, unless the patent office decides to further reduce the quality of its examinations because there is no longer a strong presumption of validity for patents that have not been "gold-plated."

Until the patent has been "gold-plated," patent owners' rights are in limbo because they must either have another examination or confirm the patent through litigation before they are sure of the scope of their rights. A patent would become simply a notice that there may be some rights that could be asserted after it has been further examined. And because the scope of the patent could change substantially when it is being further examined, any notice provided is uncertain.

In contrast, the limited patent proposed here provides immediate, but limited, rights based on registration and use in commerce. There is no need to further reduce the quality of examination to reduce the backlog of applications. And the lower application fee encourages more disclosures for the patent office's prior art database, improving the quality of both regular and limited patents when they are examined.

Copyrights, not patents, for software

In his book *Math You Can't Use*,[4] as well as two articles published in *IEEE Spectrum*,[5] Ben Klemens calls for the ending of patent protection for computer software. In his view, software should be protected only by copyright.

This proposal will certainly play well with the people who are against software patents, but experience shows that it will cause more problems than it solves. Before it became clear through a series of court decisions that software-based inventions were patentable, we had the system that Klemens proposes. For those cases where an infringer simply made a literal copy of a computer program, there was little problem. The problem came when a new program was written using techniques from an existing program. To what extent should such "non-literal" copying be an infringement?

The high-water mark in non-literal copyright protection for computer software came in *Whelan v. Jaslow*,[6] which held that the "structure, sequence, and organization" of a computer program was protected by its copyright. We don't know how far courts would have continued to stretch copyright beyond literal infringement because about the time *Whelan* was decided, the Supreme Court had found a algorithm-based invention that it felt was patentable[7] and the Federal Circuit had completed its embrace of software patents with *In re Alappat*.[8] As software patents became the preferred means for protecting a new technique, copyright reverted to protecting against the literal copying of a computer program.

[4] Brookings Institution Press, 2005, ISBN 0-8157-4942-2.

[5] "Software Patents Don't Compute," July 2005, and "New Legal Code," August 2005.

[6] 797 F.2d 1222, 230 USPQ 481 (3d Cir. 1986).

[7] *Diamond v. Diehr*, 450 U.S. 175 (1981). But software patents had issued well before then. For example, see U.S. Patent 3,568,156, "Text Matching Algorithm," granted in 1971. (The inventor, Kenneth Thompson, is also one of the creators of the Unix operating system.)

[8] 33 F.3d 1526, 31 USPQ2d 1545 (Fed. Cir. 1994).

Determining what is protected

Klemens points to the ease of getting a copyright compared to a patent. A copyright comes into being at the time of fixation of a work, and a simple registration form must be filed before an infringement suit can be brought. But such simplicity comes at a price – as cases like *Whelan* show, it is hard to determine just what is protected by a copyright, making it difficult for a person wanting to produce a new implementation of a computer program.[9] Because of the claiming requirement for patents, it is far easier to know in advance what a patent covers than what a copyright covers, especially if copyrights were to expand again to cover more and more non-literal aspects of a computer program because patent protection is not longer available.

In his book, Klemens recognizes that it might be necessary to go beyond protection for literal copying if copyright were to replace patents. "The correct breadth recalls the rule of thumb that protecting the interface is detrimental but protecting the implementation from theft is essential, but using that rule in the copyright realm requires new considerations: copyright can be interpreted too narrowly, since a program with the variable names changed is still the same program." But he gives little guidance of how that line can be drawn in practice.

More troubling, in an example he indicates that a situation where an "imitation would likely be infringing," he is troubled that applying the same test to software "becomes equivalent to a patent on an interface, and … such breadth is economically detrimental." Presumably, he would like to see copyright law for software develop on a different track from current copyright law, increasing the uncertainty about the breadth of protection until suitable case law develops and is generally accepted.

Disclosure is important

Also lost in Klemens' proposal is the disclosure requirement that forms such an important part of the patent system. Even with "open source" software, it is difficult to find how a particular function is performed unless that function is an obvious part of a known program.

In fact, since adoption of the Copyright Act of 1976, there is no longer a requirement that the protected work even be published. A trade secret, written down or other fixed in a tangible medium of expression, is protected to the same extent as a book on sale,[10] even though its protected expression is unavailable except through a trade secret agreement. This is the case for most proprietary computer software.

In contrast, a patent concentrates on one particular technique, and that technique must be described fully in the published patent, so that a skilled person can implement and use the technique without undue experimentation. The disclosure is also manually placed within a classification system so that it can be readily located.[11] The limited patent proposed in this paper would

[9] Linux is a new implementation based on Unix operating system. Many programs have been reimplemented by "free software" advocates to "liberate" them from their proprietary status.

[10] Perhaps even more, since the term of a work made for hire is 95 years from its first publication, or 120 years from the date of its creation, which ever comes first. 17 U.S.C. § 302(c).

[11] When there are too many patents within a particular class and subclass, the patent office breaks the subclass (and related subclasses) into more specific subclasses or

continue this disclosure requirement and the classification of techniques by the patent office, and would enhance it by encouraging more filings because of the lower fee and simplified registration procedure.

Independent creation as a safe harbor

But there is a reason why software developers are less concerned about copyrights than patents, including their longer term. To infringe a copyright, you have to have based your work on the copyrighted work. No matter how similar your work is to another, if you can show that you independently created your work, you are not an infringer.

There is no such safe harbor for a patent infringer. If what you are doing meets all the elements of any claim of a patent, you are an infringer. It makes no difference whether you have ever seen the patented thing or are aware of the patent. As some recent high-profile cases have shown, a software developer can plow millions into development of a new system, but can be stopped by the owner of a patent that is not even producing a product or licensing the technology to a manufacturer.

This scares most software developers, especially when the quality of some patents is considered. And that is why the limited patent proposed here provides for a “substantial completion” defense as well as a showing that the infringement is based on the patented thing.

Copyright is not the solution

Klemens spends little time on another problem with copyrights – their term of protection. Many people think the twenty-years-from-filing term for patents is far too long for computer software. Patents for Microsoft’s Windows 95 are now just expiring. But copyright lasts seventy years beyond the death of the last author or, in the case of a published work made for hire, 95 years. The copyright on Windows 95 will not expire until the end of 2090!

Eliminating software patents and going to copyright as the only protection is likely to cause new distortions in copyright. It is better to look at those aspects of copyright protection, such as the defense of independent creation, combine them with the best parts of patents (enabling disclosure and required claiming), and set an appropriate term of protection (four years, rather than 20 years for patents and 95 years or more for copyright). That is what the limited patent proposed here does.

sometimes a new classification. For example, software-based inventions were initially a subclass within the class for computers. They later became their own class. Now, they span a number of classes, with an entire class for database techniques and another for artificial intelligence.