Introduction

The recognition of patents granted to secure the profits reaped by a creative man or woman's useful, luxurious, or otherwise desirable new product to its inventor is considered to have been practiced under the influence of a governing body since the Venetian Statue of 1474, though some experts argue that this convention has been around for since the rule of ancient Greece. Regardless, patent enforcement in some regard has found its way into the public law of nearly every country of the 21st century. In these countries, patent holders are promised certain rights and monopolies over products, or more accurately ideas. The development of patent law and its divergent evolution and adoption in various parts of the world has resulted in very complex, and in some cases incompatible, implementations of patent protection. For some entities, this has opened avenues of abuse and encouraged instances of what some would call blatant overstepping of ethical boundaries in the context of interpreting original inventions, particularly in the world of software. It cannot be denied, however, that patent law has held up the integrity of authentic innovation in some capacity. Discussed here will be a defense of software patents in the United states, but also an argument put forth for the absolute abolishment of this bureaucratic function.

For

A popular argument for the upholding of software patents in the United States, which has also been used in lobbying for the recognition of software patents internationally, is that the availability of patents in an industry has a positive effect on the rate of innovation within that industry[1]. Ultimately, patents secure the right for a person or organization to profit exclusively from an original idea. While profit from a product has obvious immediate benefits for the direct profiteer – in this case the patent holder – profit has the additional manifest function of attracting investors. If the securing of new patents for in-demand products can be capitalized upon, the act of investing time and money into developing new patents becomes justly incentivized. These are the basic necessities for research and development projects. With these artifacts secured, a feedback loop dependent on the capital liabilities of the fruits of research can begin: research and development makes money for investors, so the investors allocate more resources to research and development, ad infinitum. In theory, the expected result of this process is a boost in the rate of innovation that occurs in the society that recognizes these patents and greater economic benefits provided to the parties involved.

The above point about making money applies almost universally. Revenue is a basic goal of most people in a technological society, but there is more to revenue than R&D investments. More often, a small company will develop a new algorithm to accomplish a specific task faster, more efficiently, or for the first time – it could be argued in this case that the algorithm was *invented* but that is a different philosophical debate. How does a small software company protect its innovations from being reverse engineered by the competing technology giants and replicated, effectively stealing any potential benefit that may have resulted from this engineering feat? Copyright protection is one answer; however, copyrights do not provide the same full protections to the copyright owners as a patent would. In essence, copyright registration prevents unlawful copying and repackaging or redistribution of unlicensed software[2]. What copyright protection is not able to accomplish is protecting the copyright holder from the aforementioned process of reverse engineering then reimplementing. There is no guarantee that the new implementation will reflect the original in any way, and even if it does, the act of copying must be proven. More generally, registered copyrights protect *works* that are in some sense authored or creative, rather than *inventions*, *processes*, or *ideas*[2]. A patent better serves this purpose and if an algorithm or piece of software can be described as one of those three things, or something else that holds a similar semantic value, then a patent applied is guaranteed to protect the patent owner from replication of their property by a competitor. In this way, patents protect small companies from losing exposure of original inventions to enterprise-scale clones. This has been practiced in recent history in cases where small companies, even if they have already ceased business functions, have been granted legal recognition for pioneering certain software concepts that were being infringed upon by much larger organizations[3]. In this way, the filing and recognition of

[1] Griliches (1994) studied the relationship between investment in research & development and economic turnaround.
[2] The official website of the United States Patent and Trademark Office details the key differences between patents and copyrights.
[3] Small companies have been granted remission and appeal over previously dismissed cases of patent infringement on erroneously invalidated patents, such as in the cases of *Planet Blue v. Bandai Namco Games America* and *Enfish LLC v. Microsoft*.

software patents protects small companies from ethically questionable practices that could potentially be carried out by members of the more powerful corporate opposition.

So far described as reasons in favor of software patents have been some of the benefits experiences by various entities involved in the software patent system. These are, however, idealistic. There is another way approach the defense of software patents. Logically, if software creations can be presented as having equal value as hardware creations, the patentability of which is hardly disputed, then it stands to reason that patents for software creations should be just as valid. On one stage, the amount of cooperation that occurs today between software and specialized hardware proves that software is worthy of its patent[4]. Some machines such as GPS satellites or surgical robots would not serve a function if the software for these systems was not written. If the software running the machine is just as critical to the invention and often just as complex, then why does only the machine deserve the recognition as useful innovation? The other stage is that software design has arguable become more influential than the hardware upon which the software runs. In the realm of technology, most of the everyday functions of personal technology, business operations, and government procedure are described in a human-readable language, which is transformed in such a way that is can be carried out by a general-purpose task-execution machine. The hardware, in this case, has received its patents due to filling the need for *a computer*, but it is not capable of solving these specific types of problems without additional direction that ultimately comes from a human user at some level. If this user's wit and labor are what leads to problems being solved, then they should be valued no differently than they would be if the result was hardware.

[4] Kappos (2012) spoke on the role of software in the infrastructure of a technological society such as the United States as an argument for the legitimacy of software patents.

Against

There are many faces to the cons of software patents, but the most obvious are those founded upon the glaring ambiguities in the definition of *patentability* and how software simply does not meet those criteria in some perspective. Previously, the defense was presented that software patents are legitimate due to being integral for realizing the practical problem-solving capabilities of computational hardware. It must be considered, however, that software design is not compatible with the assumptions of patent law, which is that an invention is structural in nature[5]. This is false: software exists consummately within the bounds of the structure of a computer. Because software is restricted to the limits of the structure, it does not do anything to make a computer work but rather *is the computer working*. The Supreme Court ruled in 2012 that processes that essentially amount to matters of data-gathering and novel tasks are not patentable[6]. Principally, it is difficult, if not impossible, for any software to fall outside of this description. This complements the other common issue of software patents, which is that all software is fundamentally a complex mathematical expression according to the Church-Turing thesis. Math has also been ruled as unpatentable by the Supreme Court[7]. These are serious logical and legal hurdles that must be addressed before software patents can become widely accepted.

Software patents do not just face logical issues, but also practical issues. The encouragement of software patents and the high number of them that are granted each year has brought forth the phenomenon known as the "patent thicket." As the number of software patents grows, software companies seeking to realize a new product must search through all active software patents in order to understand the potential legal ramifications of doing so. Additionally, software patents have allowed monopolies to practice "industrial gatekeeping," which, contrary to the intuitive positive effect of patents described before, effectively makes it impossible for certain organizations to pursue research in some fields[8]. Having to constantly keep up with the flow of new software patents and comparing them to one's own patent or product is not only time consuming but incredible costly. There is not even a guarantee that a valid "patentable" item will receive its patent. There have been cases where a product pending a patent entered obsolescence long before the patent was granted. This really only even applies to companies with funding. The cost of filing a software patent can reach as high as $20,000, which is well beyond the means of individuals. These singular entities must depend on and wait through all of the bureaucracy of personal patent filing organizations. In general, the software patent process has shown itself to be highly. The barrier of entry is too high for the average software developer, the cost is usually higher than the benefits received from litigation cases, the patent proposal process takes much longer than it needs to, and there is ultimately no actual guarantee that the patent will yield any results for the patent holder or protect them from patent infringement, whether done with intent or without.

---

[5] Plotkin (2002) argues that the lack of influence on the structure and mechanics of software on electromechanical machinery disqualifies it as a patentable resource.

[6] The SCOTUS ruled in *Mayo v. Prometheus* that a clinical diagnostic that amounted to two steps of data-gathering and a third unpatentable novelty step is not eligible for patenting. This case has seen widespread use as a template against which the acceptability of "soft" patents, including software patents, can be validated.

[7] The SCOTUS ruled in *Alice Corp. v. CLS Bank Int'l* that a mathematical function for computing "settlement risk" cannot be patented.

[8] Almarin (1966) describes how large organizations can abuse the patent system to eliminate competitors in R&D industries.

On top of the arguments previously offered, software patents face a massive slew of ethical issues that casts a massive shadow on the credibility of even legitimate patent holders. This begins in the patent office itself. Specifically, the examiners of software patent applications are expected to understand the content of the patent well enough to make a decision on whether or not the documented process is valid eligible to receive a patent. However, these patent application overseers more often than not will not have any exposure to the software industry. Because of this many patents that should not be granted make it through the patent office, due simply to the impossibility of ensuring that the patent examining staff are well educated in all fields from which patents could come. Importantly, this allows two types of patent applications through that should not be granted such privilege. One of these types of erroneous patent are simply that done of a software process that is considered to be trivial to the average software developer. The object being patented could potentially be found in the codebases of thousands of organizations, large and small, exposing them to avoidable litigation. The second type of patent that can come through due to innocent ignorance is the troll patent. Troll patents are those which are secured solely for the purpose of having exclusive rights to a certain area of innovation. To this end, these patents are made as abstract and vague as possible to cover wide ranges of potential interpretation. This is done by large companies to maintain a lead in development for a particular discipline within software, but it is also a maneuver carried out notoriously by Non-Practicing Entities. These people or organizations acquire software patents with no intention to apply the patented technology, but rather to open up avenues of litigation for patent infringement. This habit of different groups acquiring such a wide span of patents is hurting creativity, freedom, and progress within the greater software community – both closed and open source. In many ways, software patents have done more to hinder the advancement of software as opposed to help it.

Personal Opinion

Before, I think I was adamantly against the idea of software patents. I failed to grasp any concept of a benefit that could come out of acquiring one. To me, they seemed only useful to an end of technical abuse. Essentially, the idea of a software patent was automatically synonymous with patent trolling. This made it difficult for me to write this paper, as I had to rely on unofficial sources to tell me what kind of things make a software patent defendable. As it turns out, I had a learning moment during the research phase, and I am not so wholly against software patents as a practice.

My opinion on software patents, after this epiphany of sorts, is centered more heavily on its affects on individuals or small groups rather than the affect that its actual manifest function, e.g. patent trolling, has. For the most part, I still see software patents as something highly abuseable. From my perspective I have rarely witnessed acts of software patents defending an honest entity, or software patents being granted defensively to in-production or actively distributed software. I have only heard about this occurring once when Amazon patented their one-click checkout button. More often I see entities use software patents as a means to "squat" in an undeveloped software practice. If a granted patent is not taken with this purpose, then it usually seems to be for patent trolling. Generally, I have yet to witness software patents being used for any ethical function by those who like to hoard them.

I did say, however, that I saw some light in this institution. Mostly, the point that convinced me to defend software patents in some respect is the possibility for software patents to protect small software companies from the shady practices of large enterprises. I would like to see more cases of software patents upholding the integrity of small businesses. If this kind of defense happens to appear more often, my new faith in software patents will have been validated. This is really a waiting game, and I realize it is not guaranteed to happen. The examples that I produced in the *For* section were brought to my attention during research by others who adamantly defend software patents. Their defense make sense, but these interpretations of software patent infringement seem somewhat dubious. I think a lot of adjustment needs to come to the patent system to bring more trust to software patents. If the patent office could somehow measure the "intent to implement" of a pending patent and judge patentability upon that, I believe an increase in the quality of patent infringement cases would be observed. Specifically, pending patents for software should include a working prototype, which is much easier to do with software. The patent can be measured against the prototype to see if the patent details sensibly reflect the give software's specific function(s). If the patent is so abstract in nature that there is room for it to protect more than what the prototype can demonstrate, the patent should be rejected. Alternatively, if the prototype seems to address everything within the limits of the patent's protections, then the patent is sensibly valid: the implementation has already been proven and solved to some extent. This could reduce patent squatting and patent trolling significantly because the prosecution in a patent infringement case would need to prove that the infringing artifact is functionally equivalent to the prototype. This is just one suggestion on how the patent system may be improved. Without some change, I will continue to see software patents as mostly a dishonest through which to carry out legal abuse.

Bibliography

Alice Corp. v. CLS Bank Int'l. 2014. 13-298 (Supreme Court of the United States, June 19).

ENFISH, LLC, v. MICROSOFT CORPORATION, FISERV, INC., INTUIT, INC., SAGE SOFTWARE, INC., JACK HENRY & ASSOCIATES, INC. 2016. 2015-1244 (United States Court of Appeals for the Federal Circuit, May 12).

Griliches, Zvi. 1994. "Productivity, R&D, and the Data Constraint." *The American Economic Review* 84: 1-23.

2012. *An Examination of Software Patents.* Performed by David Kappos. Center for American Progress, Washington D.C. November 20.

MAYO COLLABORATIVE SERVICES, dba Mayo Medical Laboratories, et al., Petitioners v. PROMETHEUS LABORATORIES, INC. 2012. 10-1150 (Supreme Court of the United States, March 20).

MCRO, INC., DBA PLANET BLUE, v. BANDAI NAMCO GAMES AMERICA INC. 2016. 2015-1080 (United States Court of Appeals for the Federal Circuit, September 13).

Phillips, Almarin. 1966. "Patents, Potential Competition, and Technical Progress." *The American Economic Review* 56: 301-310.

Plotkin, R. 2002. *Intellectual property and the process of invention: why software is different.* Raleigh, NC: IEEE.

United States Patent and Trademark Office. n.d. *Trademark, Patent, or Copyright?* Accessed January 22, 2020. https://www.uspto.gov/trademarks-getting-started/trademark-basics/trademark-patent-or-copyright.