International Journal of Information Technology and Knowledge Management
January-June 2012, Volume 5, No. 1, pp. 141-145

"MANAGING INTELLECTUAL PROPERTY RIGHTS IN SOFTWARE ENGINEERING"

T. M. Kirankumar

ABSTRACT: Managing Intellectual Property is about using Intellectual Property Rights (IPRs) to protect an innovative concept in order to produce commercial advantage software. Software engineering can be viewed as IPR's from both ethical and legal perspectives. As systems are deployed into real-world use, legal issues of intellectual property rights and liability will play an increasing role in shaping the industry. This paper develops a conceptual framework for the legal issues affecting intellectual property rights and liability of software engineering products. IPR's protection legislation has carried on at a speed never before known, but in judicial practice, there are still a lot of issues worth exploring in software infringement claim.

Keywords: Intellectual Property Rights, Copyright, Patents , Liability .

1. INTRODUCTION

Information technology has changed the economics of production and distribution of information products. But some Intellectual Property Right (hereafter,IPR) problems aroused with the prosperity of information technology. Especially in the software industry, software piracy is very popular. Huge increase in claims of private ownership of software intellectual property, and intellectual property regulations apply to more types of entities, are more restrictive of what may be done with those items to which they apply, and are longer lasting than ever before. Presumably never before in the history of intellectual property rights one could have observed within such a short period of time, as a decade or so, such an enormous gain in their economic importance. Taking patents as potentially most important and relatively easy to monitor form of intellectual property rights (IPRs), their number, for instance in the United States of America (USA) doubled from 1988 to 1998 to 160,000 patents and 260,000 patent applications [1]. In the European Patent Office the number of patent applications increased from 79,000 in 1995 to 140,000 in the year 2000, i.e. by 77 percent, with upward tendency [2]. Interesting and impressive are also figures on royalties paid for patent licenses in the USA, which increased from US$3 billion in 1980 to 15 billion in 1990, only to exceed the magic mark of US$100 billion in 1997; [3] or, for instance, the fact that commodities constituted 62 percent of the market value of the manufacturing industry in the USA in 1980, but less than 30 percent in 1998. For businesses that sell products or services on cutting-edge technologies that figure, for instance in Japan, are even less than 20 percent [4] of these more than 1,000 for computer software.

[1] Assistant Professor, Department of Master of Computer Applications Siddaganga Institute of Technology, Tumkur-572103. E-mail: tmkiran@yahoo.com

Economic activity is carried out within a legal framework created and enforced by government. For private industry to invest in new technology it needs to have clearly defined ownership to make a profit. The other side of rights is responsibility - the liability incurred in using and selling the products of technology. Industry must also be able to manage the risk entailed by liability. The legal system of intellectual property rights and liabilities profoundly affects the economic viability and course of new technology.

2. INTELLECTUAL PROPERTY RIGHT

Intellectual property stems from the concept of assigning people property rights to their creations of intellect. These rights are very similar to rights in real property, such as real estate. These rights include the ability to buy, sell and license their property, the ability to use the property themselves, and the rights to prevent others from using the property without permission. From a utilitarian viewpoint, this concept is based on the economic theory that private ownership causes the most efficient use of resources in a free market system [5]. From a moral viewpoint, this concept is influenced by the Lockean notion that people are entitled to the fruit of their labor-including the labor of their minds [6]. Intellectual property rights are the building blocks for managing intellectual property they are a collection of registrable and unregistrable rights, which have different but sometimes overlapping uses. IPR includes patents, registered designs, unregistered design, Right, copyright, database right, semiconductor design right, registered trademarks, unregistered trademarks, domain names and confidential information.

(A) Copyright

Copyrights provide property rights to authors and creators of artistic works. The subject matter of copyright has been expanded to cover most all-creative works. Most important is that a copyright covers the expression of an idea but not the underlying idea itself. Copyrightable works are created by writing a story on paper, recording a song, or writing a file of source code. Without copyright, there would be no financial incentive to create works, since others could simply copy and sell the creation. The copyright arises automatically once the work is recorded on a permanent medium (paper, magnetic tape or disk drive), and no copyright statement or notice is required. A copyright is enforceable for the author's life, plus 50 years.

(B) Software Patents

Governments have long known that advancing public knowledge of technology supports the common wealth. The question is how to encourage private enterprises, who would normally keep their technological advances secret for competitive advantage, to divulge their technology to the public domain [7]. The patent system provides a tradeoff, in that if an inventor fully divulges how to practice his invention, the government will allow the inventor to exclude others from practicing his invention for a limited period. When this period of enforceable exclusion ends, everyone may Practice the invention. The inventor Exchanges secrecy for temporary rights to government backed enforcement.

A patent covers an invention, which is simply a process or apparatus for solving a problem. Patents cover ideas of practical utility, whether it is a new pharmaceutical drug or a mechanical mousetrap. However, abstract ideas and laws of nature are not patentable, as they are held to be fundamental truths or manifestations of nature, and hence free to all persons and reserved exclusively to none (Diamond v. Chakrabarty, 447 U.S. 303,309(1980)). Especially basic tools of science such as manipulations of mathematical formulae are free to all persons and hence not patentable.

3. COPYRIGHT VERSUS PATENT

The common view on copyright versus patenting is that copyright protects the 'expression' of an idea, while a patent protects the "idea itself". An 'idea' is here understood to include a perspective on a functional application [8]. An intriguing case occurs when both, orthogonal mechanisms are applicable to protect the same artifact. The first line of defense is to use copyright to protect owner of the expression (which is an embodiment of invention) against IPR violations.

The second line of defense is to patent specific ideas. Since legal procedures involving patents imply significant legal risks and associated costs, it will be preferable to use, in case of a perceived violation, the first line of defense (copyright) whenever possible.

This combined usage of copyrights/patents is only relevant for a patent holder who actually exploits the inventions described in his patents. He can only do so by producing embodiments and these are in many cases susceptible to copyright violation. A patent holder, who does not own embodiments of inventions described in the patents he owns, cannot resort to copyright protection. This implies that in standard case for which the patent system has been setup, patents and copyright protection go hand in hand.

Copyrighting is an indispensable tool because copyright violation is often easier to establish than patent infringement. However, patent protection is unavoidable in all cases where copyright protection falls short. This standard case does not seem to correspond with the current practice of software protection. Rarely, holders of software patents want to enforce a monopoly on the production and delivery of the inventions described in their patents, but instead they prefer to use copyright to protect one specific embodiment. It is this strange situation where the practice of software patenting strongly departs from Philosophical background of patent system that leads to optical illusion that copyrighting suffices. Patents are intended to create temporary monopolies, but in practice copyrights are misused to create such software monopolies. Given the long duration of copyright protection, this gives undesired and lengthy protection to pioneers in the market. The Key problem is excessive length of copyright protection in combination with the fact that copyright protection has not been designed for creating economic monopolies.

4. HISTORY OF SOFTWARE PROPERTY RIGHTS

In the early history of software development, the courts took the position that patents did not apply to software. The courts' view was that computers simply used mathematical algorithms, which if viewed as mental processes are not patentable (Gottschalk v. Benson 409 U.S. 63 (1972); in this case a method for converting BCD numerals into binary numerals was found not patentable). Therefore the only way to protect computer code was with trade secret laws and copyright.

For trade secret laws to protect your invention, you must proactively work to keep it secret. Further, your invention needs to be something not easily discoverable. Therefore, most trade secrets tend to be production methods, where competitors cannot fathom from the end product how it was made. In contrast, computer executable code is fairly easy to disassemble and comprehend. Therefore, software developers have rarely been able enforce trade secret protection. More often they sought to protect their code with copyright law.

However, copyright only protected the written expression. It did not protect any of the underlying ideas encompassed by the software. Anyone could disassemble your code, see how it worked, and then write their own code to perform the same function as your code. As time passed, people tried to stretch copyright protection to cover more than simply the written expression. This led to several of the "look and feel" cases, where courts decided that copyright protected not only the code, but the "structure, sequence, and organization" of the program

5. THE SOFTWARE LIFE CYCLE

In Software engineering, the software life cycle is a frequently used manner of organizing the software development process. Figure [1] shows a strongly simplified version of the life cycle taken from a standard textbook [9]. It consists of the following phases:

- Requirements engineering: Collect the requirements and expectations from the future owners and users of the system.
- Design: Translate the requirements in a specification that describes the global architecture and the functionality of the system.
- Implementation: Build the system, and transforming the design into software source code.
- Testing: Test that the implemented system conforms to the specification.
- Maintenance: Install, maintain and gradually improve the system.

It should be emphasized that the software life cycle covers design and construction of a software product as well as its use. Each phase contains a Validation and Verification (V&V) sub-phase in which the quality of the deliverables of that phase are controlled. Also note the backward arrows that make this into a real "cycle": it is possible to discover in later phases that decisions made in a previous phase have to be revised.

We will now proceed in three steps. First, a defensive Patent-aware Software Life Cycle is sketched that ensures that the software development organization does not infringe patents of third parties. Next, a more offensive Patent-based Software Life Cycle is described that also considers the options to file patent applications for knowledge that has been generated in each phase of the life cycle. Finally, the IPR-based Software Life Cycle extends the previous one to all IPR options: secrecy, copyrights and patents.

(A) The Patent-based Software Life Cycle

It is, however, possible to go one step further. In Figure [2] we sketch a Patent-based Software Life Cycle in which yet another sub-phase has been added that performs patent applications whenever possible. We conjecture that this strategy is only available to the software development organizations with the deepest pockets. For each phase now further questions apply, such as

- Does this phase generate patentable knowledge?
- Should we file a patent application for this knowledge?
- Are there other means to avoid that this knowledge generates an advantage for our competitors?

In many large software development organizations there exist "Chinese walls" between software developers and patent attorneys. This is not only the case for large commercial organizations but also for large open source projects like the Apache Foundation. The rationale being that the less software developers know about patents the stronger the position of the organization is in legal disputes. Implementation of the Patent-based Software Life Cycle may require similar measures. Of course, such measures completely defeat one of the primary goals of the patent system, i.e., knowledge dissemination.

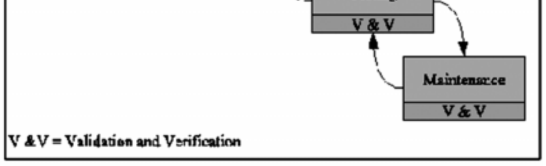

Figure 1:

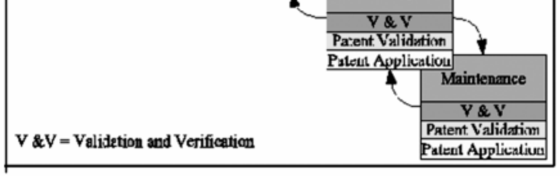

Figure 2:

These extended software life cycles already raise many fundamental questions that are not easy to answer:

- Is it possible to use these extended software life cycles in such a way that they comply with the major patenting systems worldwide?
- How can the software engineering knowledge that is hidden in the patent data bases made accessible for software engineers?

6. LIABILITY

In contrast to intellectual property rights, where the Practice of software engineering principles could lead to legal ambiguity. In particular, by contributing to the elevation of software development from an art-form to a professional engineering discipline.

The Anglo-Saxon common law (tort law) maintains that Persons who are injured by the acts of others deserve to be compensated for their injuries. A legal framework which provides clear delineation of the standards and Compensations greatly advance social and economic

Transactions. Not knowing who will be held responsible for an act is worse than knowing exactly who will be blamed. A party that knows they are at risk can do several things work to reduce those risks, transfer the risks to another party through contract, or insure against those risks. If parties do not know their risks of liability, they will over-or underinsure, or avoid the market altogether. Liability can arise through many acts, including intentional acts, negligence, and product liability. For software engineering, negligence and product liability are the most important kinds of liability.

(A) Negligence

Negligence involves unintentional but foreseeable harm.

Negligence has four basic elements: a duty of care, a breach of that duty, a causal link between that breach of duty and the injury, and an injury. Each of these elements has standards and levels of analysis, based on the act and result.

Foreseeability ties the actor's duty to potential injured Parties. Breach of duty is a positive act which the actor Performs which could foreseeable cause harm through Causal links. In the case of software writing, it could be the coding and distributing of a program that has not been fully tested, or could have side effects, which the coder should have foreseen. Injury is harm done to a person or their property. Injury is usually measured by the damage to the injured person; the purpose of compensation is to return the injured person; the purpose of compensation is to return the injured party to their state before the injury. Injury can also be economic. If a software system causes a brokerage firm to lose money through faulty advice, the brokerage may have a case. Therefore most software contractors currently explicitly state in their agreements that they will not be responsible for any loss of business resulting from use of the software system.

(B) Product Liability

Product liability has a different standard than negligence.

Product liability only applies to products, not services. However, several court cases have held that software is a product, and therefore strict liability may apply[10]. Product liability uses the theory of strict liability, which does not consider whether the vendor acted reasonably. If the product was sold in a defective or unreasonably dangerous condition, and the user was injured while using the product for its intended use, the vendor will be held liable. If a software producer develops a computer controlled braking system for a vehicle, and a bug in the software causes the brakes to fail, the producer will be held liable no matter how carefully he debugged the software.

There are many reasons for this rigid result. As described earlier, the risk needs to be assigned somewhere, and the Manufacturer seems the best choice. The manufacturer is in the best position to prevent such defects in its product. Holding it strictly liable will motivate it to thoroughly test Its products. Further, proving negligence in manufacturing is often extremely difficult and expensive for the injured Party.

An injured party may sue any person in the chain of distribution or manufacture of a product. For example, if Company A is selling software applications produced with a function library from Company B, and the application was dangerously defective due to a bug in the function library, someone who is injured may sue Company B, who produced the defective function library, or Company A, who used the defective function library in its application. However, if Company A does get sued, it may turn around and sue Company B for the damages it pays out, assuming that it can prove that the function library, not Company A's application, was defective. Also Company A would need to show it could not have reasonably found the defect during the development of its application. If Company B were found responsible, it would compensate Company A for the damages it paid to the injured party.

Today, software engineering is more of an art than a scientifically based engineering discipline. It is rare for rigorous functional specifications to be given as requirements for software applications or software components. This is in contrast to other engineering disciplines where, for example, an aircraft engine is specified in terms of a number of unambiguous parameters such as thrust, weight, fuel consumption, and maximal time between maintenance procedures. Even the bolts holding together an aircraft frame are specified in terms of weight,thread specifications, and breaking strength. Because software is typically specified in a way that is ultimately ambiguous, it can be difficult for an application developer to prove that components developed by any party earlier in the chain of manufacture were defective.

7. CONCLUSION

Engineers have always known that innovation is important, and that patents protect their inventions. Software engineers know that the programs they write automatically protected by copyright. And accountants are rapidly becoming aware of the value os all such intellectual property rights, even to extent of a few forward looking companies putting IPR'S on their balance sheet as assets. Changes to law tend to lag changes to society by a number of years. It takes time for disputes involving new concepts to work though the courts and effect changes to the law. But knowing the legal issues can greatly assist developers of cutting-edge technology. The law will evolve to shape the area, and the area will shape the law. As Software engineering systems convert software production from an art to an engineering discipline, clearer lines can be set for responsibility and liability of final products. System specifiers, domain theory producers, and the application generators themselves will be held to professional standards for quality, expectations and answerability. Understanding how liability will be determined, and what can be done to minimize or shift liability, will encourage the software industry to accept and embrace this new engineers. In this paper, I have outlined some of the important open issues in managing IPR'S for software engineering, as we engineer critical systems for the next millennium.

REFERENCES

[1] Rivette and Kline, Rembrandts in the Attic-Unlocking the Hidden Value of Patents, Boston 2000, pp. 4.

[2] According to the Recently Published Statistics of the European Patent Office.

[3] Cf. Berman, "The Emergence of an Invisible" Asset Class, in: Berman, Hidden Value: Profiting from the Intellectual Property Economy, London 1999, pp. 12; cf. Also Rivette and Kline, op. cit., pp. 6.

[4] Kondo, "Roles of the Intellectual Property Rights System in Economic Development in the Light of Japanese Economy", AIPPI Journal of the Japanese Group January 2000, pp. 28.

[5] R. Posner, Economic Analysis of Law, Fourth Edition, Little, Brown, and Company, Boston, MA, 1992.

[6] J. Hughes, "The Philosophy of Intellectual Property", Georgetown Law Journal 77, No. 2, December 1988,PP 287-366.

[7] E. Kitch and H. Perhan, "Legal Regulation of the Competive Process", The Foundation Press, NY, NY, 1989, pp. 365-902

[8] Bergstra J A and Klint P, About 'Trival' Software Patents: The Isnot Case, Science of Computer Programming , 16 (3) (2007) 264-285.

[9] H. van Vliet , Software Engineering: Principles and Practice. Wiley, Decond Edition, 2000.

[10] L. Levy and S. Bell, "Software Product Liability: Understanding and Minimizing the Risks", Boa1t Hall School of Law Review, Spring 1990.