Patent Issues Aplenty

An review of software patent problems

Stuart Bryson, #98082365

University of Technology, Sydney

ABSTRACT 3

INTRODUCTION 3

INTELLECTUAL PROPERTY 3
- Copyright 4
- Patents 4
- Software Patents 4
- Differences between Copyright and Patents 5

PATENT LEGAL DEFINITIONS AND THEIR INHERENT ISSUES 6
- The Technical Effect 6
- The Physical Process 7
- The Algorithm 9
- Other Patent Requirements 9

FURTHER PATENT ISSUES 10
- Prior Art 10
- Ambiguous Legal Definitions 10
- Patent Offices Lacking 10
- Inescapable Infringement 11
- Cost 11

CONCLUSION 12

BIBLIOGRAPHY 13

Abstract

For the last 20 years, we have seen a significant increase in the number of software patents granted. This literature review gives a brief comparison of software patents and copyright, of the fundamental issues with software patents, and recommendations on how software patents should be improved.

Introduction

Just as patents are being issued in increasing numbers, so too are the issues with patents increasing. Software patents and patents in general are an important part of protecting an inventors ideas and their investment. Without software patents, we may see a decline in research and investment in software development. However, the current state of software patents is causing much debate amongst industry experts. Some are arguing to abolish software patents all together (Klemens 2005b, p59, Holmes 2000, p34) and yet others are claiming that software is no different from any other technology and can be patented (Nixon and Davidson 1997, pp.21-22).

This review will provide a survey of recent debate regarding the issues with software patents. Firstly, I discuss different types of intellectual property and give a comparison of Copyright and Patents. Following this I discuss the main legal concepts of patents and their inherent problems. Lastly, I give a summary of some further issues with software patents and conclude with some recommendations to resolve the software patent problem.

Intellectual Property

According to IP Australia, the Australian Government agency responsible for the administration of intellectual property, “intellectual property (IP) represents the product of [the] mind or intellect” (IP Australia 2005a, p2). IP encompasses different types of protection schemes including Copyright, Patents, Trade Marks and more. IP Australia claim that these schemes “provide protection for your creativity or innovation in the marketplace”.

Each scheme provides its own benefits and brings its own disadvantages. Following is a discussion a few of the more prominent schemes.

Copyright

The United States Patent and Trademark Office define copyright as:

> *"... a form of protection provided to the authors of 'original works of authorship' including literary, dramatic, musical, artistic, and certain other intellectual works, both published and unpublished." (United States Patent and Trademark Office 2005, p1)*

Australia has a similar definition of copyright, definied in the Copyright Act 1968. While copyright protects artistic works such as dramatic and musical, it also includes "source code, executable code and databanks and tables" (IP Australia 2005b, p3). Countries may differ in the fine print of copyright however most aim to protect the author of original work.

Patents

IP Australia defines a patent as "an intellectual property right granted for new technology you have invented" (IP Australia 2005a, p2). Further, the owner of the patent is able to use the patent to their commercial advantage for the duration of the patent.

Patents have been used for over 200 years to protect the rights of inventors (Nixon & Davidson 1997, p21). The aim of patents was to "promote the progress of science and useful arts" (US Constitution Article 1, Section 8, Clause 8). This protection provided a short-term monopoly of the technology to the patent owner in exchange for public disclosure. The motivation for the inventor was to make known to the public his invention with the promise of commercial protection for a certain period (20 years). US Congress believed this would spur on innovation as it provided commercial incentives, and through public disclosure, it would promote the progress of science and useful arts.

Software Patents

A software patent behaves much the same way as traditional patents. It grants the patent owner commercial benefits while benefiting the public through disclosure of the software techniques.

176 countries around the world grant patent protection for invention. More than half of these "permit patenting of software-related inventions" (Fenwick & West LLP 2004, p5). Fenwick & West go on to explain that there is a worldwide trend towards adopting software patents. Due to this increasing interest in software patents, it is reasonable for the issues with software patents be addressed. In order to understand the issues we must outline some of the important concepts related to patents.

Differences between Copyright and Patents

While both Copyright and Patents are techniques for protecting IP, their legal definitions are quite different.

Fenwick & West explains that in general, "copyright laws protect the form of expression of an idea, but not the idea itself" (Fenwick & West 2004, p2). This is just as true for software as any other work. It "protects the specific code … and does not protect the ideas or methodologies" (IP Australia 2005b, p4). Conversely, patents will protect not only the 'form of expression of an idea', but also the idea itself. In fact, patents even provide protection against another coming up with the same idea independently (Fenwick & West 2004, p6). As patents enable the protection of an idea, "patents offer a significantly stronger form of protection for software" (IP Australia 2005b, p1).

Under Australian law, software is automatically protected by copyright. The same software, however, may be eligible to "gain patent protection in addition to protection under copyright" (IP Australia 2005b, p1). Klemens explains that "Because [copyright] doesn't offer a patent's monopoly protection, a copyright is, in some ways, weaker protection than a patent" (Klemens 2005a, 62). Bragg concurs, claiming that patents provide a "stronger, more fundamental form of protection because they protect the idea and not the source code per se". Others claim that copyright offers "scant protection" and does not protect the "underlying functionality" (Skulikaris 1999, p109).

Due to this stronger protection, some argue that patents are the most important protection of IP available. Bragg quotes Kuester as saying "Patents, not copyrights, are now the only way to give adequate protection to the most important aspects of software" (Bragg 2001, p39).

Another important distinction between patents and copyright is how they are obtained. While copyright is automatically granted, patents are not. IP Australia explains that a patent application must be filed whereby "IP Australia's examiners will assess to see whether your invention can be patented" (IP Australia 2005a, p6). The United States has a similar process through the Patent and Trademark Office. Just how patent applications are assessed is discussed in the following section.

Patent Legal Definitions and their Inherent Issues

Australia, the United States and the European Union all have similar requirements in order for a software patent application be approved, however their legal terms may differ slightly. Detailing every country's requirements is beyond the scope of this review, however, following is a culmination of the general trend around the world for software patents and their inherent problems.

One requirement for a patent is that it be a "manner of manufacture" (IP Australia 2005a, p5). This requirement aims at the prevention of the patenting of artistic creations, mathematical models, algorithms, theories, laws of nature etc. I believe that this manner of manufacture has significant overlap with the European Patent Office's doctrine on "technical effect" and further overlaps with the United States Patent and Trademark Office's guidelines released in the early 1990's on a "physical process". This is perhaps the biggest issue in the software patents debate. In fact very few authors take issue with any other requirement of software patents.

The Technical Effect

A doctrine issued by the European Patent Office aims at limiting the scope of software patents by requiring the software to cause a 'technical effect'. Fenwick & West argue that this doctrine is "the most widely followed doctrine governing the scope of patent protection for software-related inventions". Just what a technical effect is, however, is the subject of much debate.

Some have described this technical effect as “an operation that goes beyond the mere interaction between the software and a computer” (Skulikaris 1999, p110). Skulikaris, however, admits that even this definition is ‘quite vague’. He claims that the reason for this is that the European Patent Offices uses fair reasoning to establish if the patent contains a ‘technical effect’. One could assume therefore, that even the EPO does not conclusively define just what a ‘technical effect’ is.

Other authors explain that a ‘technical effect’ excludes business methods (McLaughlin 2004, p101) and e-commerce technologies (McLaughlin 2004, p102). And further still, a ‘technical effect’ is “regarded as overcoming the restriction to a computer program as such” (Blakemore 1999, p159).

The Physical Process

The United States Patent and Trademark Office’s guidelines “directed patent examiners to consider a software-related invention patentable if it was used to control an external process… or if it was fundamentally connected to a process or machine” (Bragg 2001, p.39). This has also been described as “software that affects a physical process” (Fenwick & West 2004, p.5).

Bragg expresses his concern that “a very fine line may exist between an unpatentable algorithm, technique or method and its embodiment in patented software”. It is possible to make any unpatentable algorithm patentable by simply adding it to an uninventive physical component and the whole becomes patentable (Kelmens 2005, p57). Further, some authors claim “experienced patent lawyers can always translate any claim for a series of steps into a claim for a system indistinguishable in substance from the steps claim” (Stern 1990, p10). In other words, even if the software does not affect a physical process, as the definition is so ambiguous, it could still be successfully argued that it does.

Further still to the absurdity of the ‘physical process’ requirement, some authors claim that there is indeed no distinction between hardware and software (Nixon and Davidson 1997, p22), implying therefore, that this requirement for a physical process is futile.

Plotkin claims that software is different to conventional patent claims, which focused on structure and physical features (Plotkin 2002, pp237-238). He explains that the importance of physical structure within patent law, as it "limits patent protection to products and processes that achieve 'concrete and tangible results'" (Plotkin 2002, p238). Further, he maintains that as patent law was traditionally used for other fields, such as electro-mechanics, these patents are required to be "described and claimed in terms of their physical structure" (Plotkin 2002, p238).

Plotkin presents another interesting aspect to the physical process debate. That is, software is almost always designed in terms of its function and not its physical structure. In fact, software patents rarely describe any novel physical structure but rather the claims describe correctly, that the software can be implemented on generic computer architectures. Further, the generic PC is replacing many custom-designed hardware in may different industries, improving speed and cost of manufacturing. (Plotkin 2002, p241). With the introduction of a generic PC, patent examiners will believe that this software has the appropriate physical process and approve the patent. He then explains that this will provide "too much protection, leaving insufficient room for the development of competing designs which would potentially further the 'progress of [the] useful arts'" (Plotkin 2002, p241).

And Plotkin is not the only one to have concern over the hardware / software distinction. Holmes quotes Gimlan saying "in practice software cannot be distinguished strictly from hardware" (Homes 2000, p31).

One last observation on the physical process issue, is that since software is "designed solely in terms of its function" (Plotkin 2002, p238), it is the computer itself and not the author of the software which creates the physical process. That is, a programmer will create the source code and the computer will "modif[y] its own physical structure to perform the functions described in the source code" (Plotkin 2002, p239). "What happens inside a computer during program execution is an entirely physical process" (Nixon and Davidson 1997, p22). Nichols presents another angle to this point, and explains that software has the ability to modify itself using languages such as Smalltalk (Nichols 1999, p27). If the software is modifying itself, how can one predict the resulting physical process? Indeed, this modification of self puts software in a very unique position and consequently "there [is no] adequate description in the formal language of patent specifications" (Nichols 1999, p28) to deal with software.

The Algorithm

Another important aspect of patent claims is that the technology is not an algorithm, mathematical model or physical law. Nixon & Davidson claim that many are needlessly confused about what exactly is being patented with software. The common belief is that some have tried to patent algorithms and mathematical formulae. Nixon and Davidson, however, claim it highly unusual for sketches, flow-charts, mental steps, or mathematical formulae per se, to be patented (Nixon and Davidson 1997, p22). In fact, they believe there should be "no confusion between claiming a mathematical formula per se (which is invalid), and merely using a formula as a convenient linguistic technique (which is valid)". But if there should be no confusion, why is this one of the biggest issues in software patent law to date.

While Nixon and Davidson claim this is not a problem, other authors disagree. They claim that the exclusion of algorithms is 'philosophically agreeable' but it is difficult "differentiating between a patent *for* the excluded subject matter and a patent for the *use* of the excluded subject matter" (Stratton 1994, p592).

Klemens claims that the problem has arisen as mathematics has become more dependent on computers (Klemens 2005b, p56), and that "the line between mathematical algorithms and machinery is increasingly blurred". He claims that by simply adding the generic PC, to fulfil the physical process requirement, enables the use of "any inventive mathematical algorithm" in a patent. Klemens goes on to say that software and mathematics are indeed the one and the same and that a software patent is indeed a patent on mathematics (Klemens 2005b, p59).

Others have taken a similar position, explaining that software has "attributes that make it appear no more than a method of mathematical calculation" (Blakemore 1999, p158).

Other Patent Requirements

Other requirements for software patents are that the software must be non-obvious or an inventive step (IP Australia 2005a, p5). They must also be useful "in the sense that the invention must have practical application" (Blakemore 1999, p160).

Further Patent Issues

Apart from the inherent issues found in the legal patent definitions themselves, there are also some common issues with software patents in general. Following is a description of some of the more common problems.

Prior Art

In order for the patent office to grant a patent, the technology needs to be carefully weighed against previous inventions in the field. These previous inventions are also known as "prior art". This common argument against the patentability of software is that since software patents are relatively new, beginning in the U.S. in 1981 (Webber and Cave 1997, p115), there is very little prior art documented. Stern explains that it is "harder to locate well-indexed textbooks for searching out prior-art software than for prior-art electronic circuitry … and other older fields of art" (Stern 1990, p8). Others agree claiming that patent offices do not have access to material that should have been collected over the last 30 years. (Webber and Cave 1997, p115, Stratton 1994, p593).

Some claim that one of the benefits, among others, of having international organizations for handling patent applications, such as the European Patent Office, is that they facilitate a much further reach or knowledge of prior art (Skulikaris 1999, p109).

Ambiguous Legal Definitions

Another common argument put forward against the use of software patents relates to the current state of the legal definitions. Indeed, there is a "lack of clear legal definition of patentability in this field" (Stern 1990, p9). Any attempt to discriminate clearly the patentable from the unpatentable is simply capricious (Klemens 2005b, p58). In fact, in 2002, the European Commission claimed that software patents were "ambiguous because member of states interpret patent law differently from one another" (Chatterjee 2004, p63). Holmes claims that having such ambiguous laws that ordinary lawyers cannot understand, let alone the public, is 'unethical and dangerous' (Holmes 2000, p31).

Patent Offices Lacking

Patent offices are required to review patent applications against the legal guidelines set out by the nations government. Unfortunately, most patent offices are lacking in the required

resources, be it experts in software, prior art listings, or enough time and personnel to review fully each application (Stern 1990, p8, Stratton 1994, p593, McLaughlin 2004, p102). Holmes explains that lawyers and not software professionals are responsible for the outcome of patent claims. Consequently, the incorrect decisions made by lawyers are setting invalid precedents for future claims (Holmes 2000, p31).

An interesting point to note is that IP Australia is not in fact required to check patent applications against all the requirements for a patent to be approved (IP Australia 2005a, p5). This would no doubt result in patents being awarded that should not be, possibly resulting in legal battles and claims that could have other wise been avoided.

Inescapable Infringement

Patent infringement is an issue that affects all software developers. Software developers need to ensure that they are not writing software that infringes upon any existing software patent, or indeed future patents. Unfortunately, this is usually beyond the resource limits of smaller companies and open-source developers. In fact, Hayes argues that it is "impossible to be sure that software doesn't infringe a patent" (Hayes 2005, p52). This is due to not only keeping abreast of the vast amount of software patents, but that "patents have been issued in even the most basic fields of computing" (Klemens 2005a, p60).

One solution to this for bigger companies such as IBM and Microsoft, is to hoard a vast collection of software patents and cross-license them (Bragg 2001, p41). While patent stockpiles and cross-licensing is outside the scope of this review, they do provide an oasis for some. The open source community and small developers may not have this luxury.

Cost

The cost of both acquiring software patents and staying abreast of the current patents so as not to infringe them is colossal. It is so expensive in fact that small companies and open source developers cannot generally afford either. Patent law "imposes economic costs, most notably, everyone in the industry must spend money on remaining abreast of every relevant patent. When 'industry' means everyone with a computer, that's an astronomical sum" (Klemens 2005a, p61). Acquiring a patent "for any invention, including software, is relatively expensive" (Fenwick & West 2004, p5). Bragg exclaims that with so many patents granted each year, ever-changing patent law, searching for prior art, and administration costs, means

that the cost of acquiring a patent is fast “approaching five figures per patent” (Bragg 2001, p41).

Conclusion

I have presented an introduction into the current issues surrounding software patents. I have outlined the role they play in protecting intellectual property. I have shown the inherent issues with the legal definitions such as the ‘technical effect’ doctrine and the ‘physical process’, and also discussed various other patent requirements. I have also outlined some of the other major issues with software patents such as the lack of ‘prior art’, the ambiguity in the above legal definitions, the under-resourced patent offices and the vast cost associated in obtaining and staying abreast of current patents.

In order for software patents to be effective in the future, the above issues must be resolved. Some authors have suggested instituting better prior art methods (Stern 1990, p10, Bragg 2001, p42). Others have suggested better skilled patent offices including the employment of software experts and computer scientists (Stern 1990, p11, Plotkin 2002, p242). I recommend employing these ideas and also ensuring that no software patent that is too broad is granted, and that the term of software patents are shorted from 20 years. The software patent problem must be resolved, and until it is, every patent granted will continue to highlight the inadequacies in the software patent system.

Bibliography

IP Australia, 2005a, 'Patent Guide', pp. 1-24 <www.ipaustralia.gov.au >.

IP Australia, 2005b, 'Patents for Computer Related Inventions', pp. 1-4 <www.ipaustralia.gov.au >.

Blakemore, F. 1999, 'Patenting Computer Software', *Engineering Management*, vol. 9, no. 4, pp. 157-160.

Bragg, A.W. 2001, 'Software Patents', *IT Professional*, vol. 3, no. 4, pp. 39-42.

Chatterjee, A.C. 2004, 'Europe Struggles over Software Patents', *IEEE Spectrum*, vol. 41, no. 9, pp. 61-63.

Fitzgerald, A. 1997, 'Background to Software Patenting in Australia', *Australian Software Engineering Conference*, pp. 113-114.

Hayes, F. 2005, 'Patently Fair', *Computerworld*, vol. 39, no. 17, p. 52.

Holmes, W.N. 2000, 'The Evitability of Software Patents', *Computer*, vol. 33, no. 3, pp. 30-34.

Klemens, B. 2005a, 'Invention: New Legal Code', *IEEE Spectrum*, vol. 42, no. 8, pp. 60-62.

Klemens, B. 2005b, 'Software Patents Don't Compute', *IEEE Spectrum*, vol. 42, no. 7, pp. 56-58.

LLP, F.W. 2004, '2004 Report on International Legal Protection for Computer Software', *Computer & Internet Lawyer*, vol. 21, no. 4, pp. 1-7.

McLaughlin, L. 2004, 'European Union Struggles with New Rules for Software Patents', *IEEE Software*, vol. 21, no. 5, pp. 101-104.

McLaughlin, L. 2005, 'Inside the software patents debate: some good news for open source developers', *IEEE Spectrum*, vol. 22, no. 3, pp. 102-104.

Nichols, K. 1999, 'The Age of Software Patents', *Computer*, vol. 32, no. 4, pp. 25-31.

Nixon, L.S. & Davidson, J.S. 1997, 'Software can be Patented', *IEEE Potentials*, vol. 16, no. 4, pp. 21-22.

Plotkin, R. 2002, 'Intellectual property and the process of invention: why software is different', *Technology and Society*, pp. 236-243.

Skulikaris, Y. 1999, 'European Software Patents', *IEEE Software*, vol. 16, no. 6, pp. 109-111.

Stern, R.H. 1990, 'Software Patents', *IEEE Micro*, vol. 10, no. 2, pp. 8-11.

Stratton, R.P. 1994, 'The Software Patent Problem', *Electrical and Computer Engineering*, vol. 2, IEEE, Canada, pp. 592-594.

Webber, D.B. 1997, 'Software Patents', *Australian Software Engineering*, p. 115.