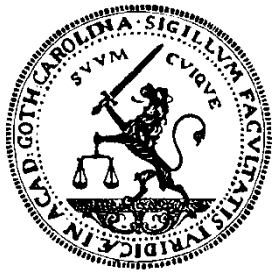

FACULTY OF LAW
Lund University

Hannes Westermann

How to treat software in the intellectual property framework

LAGF03 Essay in Legal Science

Bachelor Thesis, Master of Laws programme
15 higher education credits

Supervisor: Uta Bindreiter

Term: Spring Term 2016

Contents

Summary

The software industry is today one of the most important industries, and almost all devices we use contain some piece of software. As software has both a literal expression and an inventive aspect, it has been unclear which way the intellectual property should be protected. Software can be covered both by patents, that protect the idea behind the code, and copyright, that protects the expression of this idea.

A copyrighted program can be examined and the ideas can be reimplemented in a competitors own “words”, thereby circumventing the copyright restriction. Therefore, many have argued for the use of patents. Patents grant an exclusive right to the inventor to use the invention for 20 years, if certain conditions are met (utility, novelty, inventive step). This gives the inventor a head start in commercializing his invention and hopefully allows him to recoup the investment costs. The patent also acts as a knowledge dissemination tool. After the patent has been granted, the claim describing precisely how to implement the invention is made public. However, patents carry the risk of impeding innovation in the industry by granting a too large monopoly to one entity, preventing competition and continued development in that area.

In the United States, patent protection of software has gone through several periods. In the 70s and 80s, three cases established a very narrow patent eligibility of software. It was only patentable together with a specific machine or a process that transformed matter into another form. In the 90s, case law opened for the patenting of basically all software, as long as it was useful. This lead to the granting of hundreds of thousands of software patents. Recently, courts have gone back to not treating software as an invention, only allowing patenting if the implementation is inventive.

The view how to treat software in legal doctrine is very varied. Some argue that software is fundamentally different from hardware and should therefore not be patent eligible. They also argue that software patents have harmed the software industry. Others argue that software should be patent eligible in order to increase research and innovation. Yet others argue that a new kind of protection should be introduced for software.

I find that software is not fundamentally different from other inventions, and that it should therefore be patent eligible, as long as it meets the requirements for patentability. Many of the court cases should have been decided by applying these criteria instead of excluding software as such. The problems in practice also seem to stem from a misapplication of the inventive step and novelty tests.

However, the patent term should be decreased for software patents. The current term reflects an innovation pace unsuitable for the software industry. This would solve many of the practical problems of software patenting.

Sammanfattning

Mjukvaruindustrin är idag en av de viktigaste industrierna i världen, och flerparten av de apparater vi använder innehåller mjukvara. Eftersom mjukvara har både en litterär del och en uppfinningsrik del har det varit oklart hur mjukvara ska skyddas. Den kan skyddas av patent, som täcker idén bakom programmet, och upphovsrätt, som täcker uttrycket av idén.

Ett program endast skyddat av upphovsrätt kan undersökas och idéerna sedan återskapas med ”andra ord” av en konkurrent. Därmed kringgås upphovsrätten. Därför har många argumenterat för att använda patent på mjukvara. Patent ger en exklusiv rätt för uppfinnaren att använda en uppfinning så länge den uppfyller vissa krav (användbarhet, nyhet, uppfinningshöjd). Detta ger uppfinnaren ett försprång i kommersialiseringen av uppfinningen och en möjlighet att återvinna investeringar. Patentet fungerar även som ett sätt att sprida information. Efter att patentet meddelas blir informationen om hur man implementerar uppfinningen offentlig. Patent löper dock även risken att hindra innovation genom att ge ett för stort monopol till en enhet, vilket hindrar konkurrens och vidareutveckling i ett visst område.

I USA har patentskydd av mjukvara genomgått ett flertal perioder. På 70- och 80-talet etablerade tre rättsfall en mycket snäv patenterbarhet av mjukvara. Den var endast möjlig att patentera tillsammans med en viss apparat eller en process som förändrar materia. På 90-talet öppnades möjligheten för patentering av nästan all mjukvara, så länge den är användbar. Detta ledde till meddelandet av hundratusentals patent på mjukvara. På den senaste tiden verkar domstolen ha gått tillbaka till att behandla mjukvara som icke-patenterbar och istället bedöma om implementeringen är uppfinningsrik.

Åsikterna i doktrinen är väldigt varierad. Vissa argumenterar för att mjukvara är fundamentalt annorlunda från andra uppfinningar och därför inte bör kunna patenteras. De anser även att mjukvarupatent har skadat industrin. Andra anser att mjukvara borde vara patenterbart för att öka forskning och innovation. Andra anser att ett helt nytt skydd borde införas för mjukvara.

Jag anser inte att mjukvara är fundamentalt olikt från andra uppfinningar och att den därför borde vara patenterbar, om den uppfyller de andra kraven för patenterbarhet. Många av rättsfallen borde ha avgjorts med användningen av dessa krav istället för att utesluta mjukvaran. De praktiska problemen med mjukvarupatent verkar komma från felanvändningen av kraven för nyhet och uppfinningshöjd.

Längden för patent på mjukvara borde dock förkortas. Den nuvarande längden på 20 år reflekterar en innovationshastighet som är opassande för mjukvarubranschen. Detta skulle lösa många av de praktiska problemen.

Preface

I have long been interested in the phenomenon of software patenting. Therefore, I was very happy that I got the opportunity to examine and analyze the subject in this thesis.

Before I began my research, I was of the opinion that software patents are wrong and a senseless burden on the industry. Why should a company be sued for using common building blocks of programming? However, during the course of my writing, I realized that software patenting, if applied correctly, is a reasonable way to protect innovation in the software field. I do however believe the patent term of 20 years to be too long for the highly volatile and innovative software industry.

A big thank you goes to Uta Bindreiter, my supervisor, who with poignant questions and advice helped me find the right path.

A big thank you also goes to my father Dirk Westermann, who with invaluable advice and support helped me make this essay what it is.

I would also like to thank Martina Borgström for proofreading.

I hope you enjoy the reading!

Hannes Westermann

Abbreviations

CD	Compact Disc
DVD	Digital Versatile Disc
EPC	European Patent Convention
ICT	Information and Communication Technology
PHP	PHP: Hyptertext Preprocessor
PTO	Patent and Trademark Office
TRIPS	Agreement on Trade-Related Aspects of Intellectual Property Rights
TV	Television
US	United States
USC	United States Code
USPTO	United States Patent and Trademark Office
WIPO	World Intellectual Property Organization
WTO	World Trade Organization

1 Introduction

1.1 Background

Software is today one of the key areas of innovation. Companies spend billions developing new software products and solutions. For example, a Chevy Volt electrical vehicle contains 10 million lines of code. A recent estimate suggests 29 million people are ICT-skilled workers. Each of us interacts with tremendously complex software every day – through our smartphones, when we use our computers, when we pay by credit card, when we pay to take the bus somewhere and when we operate our TV.

The industry is struggling with how best to protect the huge investments required to create these products. The software is usually protected by copyright, which means that the author is able to prohibit competitors from using the code. However, there is nothing that prevents a competitor from buying the product and using the incorporated algorithms, ideas and designs in their own products, without using the same code. Thus, solutions developed for millions of dollars can be imitated and incorporated into a competitor's product immediately, giving an unfair advantage to the second-mover and potentially stifling innovation.

Patents give a monopoly on an idea to the inventor. He can prevent others from using the invention, which is thought to increase investment and innovation, while also motivating the inventor to explain his idea to the public. The backside is that the public cannot use and improve on the idea during the protection term. The system has to be carefully balanced to achieve a net gain for the economy.

Many have advocated the use of software patents. These do not only protect the code itself from being copied but also the idea behind the code. If someone were to patent the word processor, for example, no one else would be able to create even a similar product. As we will see later, this form of patenting is possible under certain requirements in the United States.

Many are against the use of software patents. The free software movement, which develops free and open-source software, is afraid of large companies using patents to prevent the distribution of these products. There has also been the issue of very broad software patents being granted and exploited by companies to stifle competition or gain profits through licensing.

In this thesis, I will examine the views on software patents in the United States of America. I will then examine the arguments brought up in the courts and doctrine to determine what is the proper way to treat software within the patent framework.

1.2 Purpose

My purpose is to examine the system for protecting software through patents in the United States.

1.3 Problem

The main question I will answer is the following:

From a legal and economical viewpoint, what is the proper way to treat software within the intellectual property protection system?

To accomplish this, I will look at these questions:

- What requirements does software have to fulfil in order to be protected under U.S. patent law?
- Which arguments have been brought up in legal doctrine in favour and against software patents?
- Should software be patent eligible?
- Should another form of protection be created for software?

1.4 Delimitations

This thesis will not handle questions related to:

- The utility, novelty or inventive step requirement of patents.
- Copyright.
- The patentability of business methods.
- Patentability of software outside of the United States.

1.5 Method

I will use a legal development perspective to establish the current legal situation under U.S. law, as well as the argumentations for the positions. I will then take a philosophical and a law and economics perspective in order to determine the best way to treat software in the intellectual property protection system.

1.6 Material

The material I have used is mostly case law from the United States. I have focused on cases that have had an important influence on the patent eligibility requirement. I have also used legal articles to highlight some of the viewpoints on software patentability prevalent in legal literature.

1.7 Outline

The second chapter serves as an introduction to some of the key concept related to software in general, as well as the purpose and structure of patent regulations. I will also briefly elaborate how copyrights can apply to software.

The third chapter will analyse the historical development of software patenting in the United States as well as the reasoning of the courts.

In the fourth chapter I will highlight some of the doctrinal views on the software patent debate.

In the fifth and final chapter, I will analyse the argumentation in the preceding chapters in order to find a proper way to handle the protection of software.

2 Key Concepts

Software is not one entity, but can be divided into several parts. Each of these carries their own significance in the protection of the software. Here, I will briefly explain some concepts that are important to grasp to understand the issue of software protection. I will also provide an overview of patent and copyright protection and the difference between the two.

2.1 Definitions

2.1.1 Algorithm

The algorithm is the idea contained in a piece of software. It is defined as "any detailed sequence of actions intended to perform a specific task"[1]. An algorithm is not necessarily fixed in a computer program but often written as a sequence of steps on a piece of paper, like a flowchart. By following these steps, information can be transformed in some way. This can be done either by humans or by computers. An example of an algorithm is a method to sort a list of numbers. Algorithms often have several uses, for example a control algorithm can be used to stabilize a drone or to land a rocket.

2.1.2 Source code

In order for the algorithm to be useful, the programmer has to translate it into source code. This is a text written in a programming language, such as C, PHP or JavaScript. The programmer is free to build the program however he likes, there are often several ways to solve a problem.

2.1.3 Machine code

The computer is not able to directly understand the source code. Instead, it has to be translated into machine code, a series of characters the computer can understand. This is handled by a special program known as the compiler.

2.1.4 Software

The machine code comprises the software. It is thus a collection of instructions to a processing unit. Examples of software include Microsoft Word, Microsoft Windows, Adobe Photoshop, the firmware for an iPod and

[1] Commission of the European Communities, Proposal for a Directive of the European Parliament and of the Council on the patentability of computer-implemented inventions, COM (2002) 92 final, Explanatory memorandum, p. 21.

the control software that runs on most modern cars. A commonly used definition of software for legal purposes is the following:

> *A set of instructions capable, when incorporated in a machine readable medium of causing a machine having information processing capabilities to indicate, perform or achieve a particular function, task or result.*[2]

2.1.5 Hardware

Software by itself is useless. It needs hardware to run on. This hardware can for example be a computer, a DVD-player, a car, a TV or a smartphone. Almost all electronic products today contain some sort of software to enable the functions.

2.2 Patents

2.2.1 Structure of patents

Patent protections today exists in almost all countries. A minimum level is set by the TRIPS agreement, which all 162 members of the WTO have joined. TRIPS requires members to provide patents for all inventions, whether they are products or processes. There are four requirements for patentability: There has to be an *invention (1)* that is *new (2),* involves an *inventive step (3)* and can be *applied industrially (4).*[3]

(1) The requirement of invention is usually that it is a technical creation, and not merely a discovery of something that already exists.[4] As we will see later, this has huge implications for the patent eligibility of computer programs, as computer programs "as such" are usually not seen as inventions.
(2) The requirement of novelty means that the invention has never been published anywhere in the world before the date of the application.[5]
(3) The requirement of an inventive step signifies that the invention cannot be obvious to a fictional person that is skilled in the state of the art in the field of the invention.[6]
(4) The requirement of industrial application means that the invention has to have a potential use in the industry. It has to solve a technical problem reproducibly.[7] In American law, this is known as the utility requirement.

---

[2] WIPO Model Provisions on the protection of computer software, Geneva 1978, Article 1.
[3] TRIPS, Article 27(1).
[4] Ulf Maunsbach & Ulrika Wennersten, *Grundläggande immaterialrätt* (2 ed. 2011) at 178.
[5] *Id*. at 181.
[6] *Id*. at 184.
[7] *Id*. at 180.

These steps are similar in all jurisdictions.

In order to receive this protection, the inventor has to apply for a patent at a patent office by handing in a patent application with a so called patent claim, i.e. the description of the subject to be protected, sufficiently clear to allow a person skilled in the art to carry out the invention.[8] If the patent is granted, it gives the inventor the right to prevent others from making, using, selling or even importing the invention.[9] The protection term can be extended to up to 20 years.[10] It is territorial, a new patent has to be applied for in each country where protection is desired. However, there exist treaties that simplify this process, such as the European Patent Convention (EPC).

2.2.2 Purpose of patents

There are several main purposes to patent protection. The first is to grant the inventor an incentive to innovate. The fear is that competitors can steal an idea and incorporate it into their own product, that they then can sell without having to recoup the cost of development. This means they could sell the product cheaper, outcompeting the original inventor and making innovation unprofitable. The patent grants the inventor the exclusive right to use the idea for up to 20 years, giving adequate time to recoup the costs before competitors can enter the market.[11]

Of course, this exclusive right also shuts out other parties from applying and innovating on the idea. This can decrease innovation and essentially freeze an entire industry sector for the duration of the patent term, if the patent is important.

Another important purpose is the dissemination of ideas. Without patents, inventors would have no incentive to publicize their findings, as this would risk imitators. Instead, they would be incentivized to hide the invention and how to carry it out. Many revolutionary ideas would potentially never reach the public and other inventors. Patents can be seen as a reward from the state for making the invention available to the public in an understandable way, making other inventors able to use and improve upon the design after the exclusivity period has ended and thus advancing the state of research.[12]

[8] TRIPS, Article 29(1).
[9] TRIPS, Article 33.
[10] TRIPS, Article 33.
[11] Maunsbach, *supra*, at 168.
[12] Id.

2.3 Copyright

Another form of protection for intellectual property is copyright. Copyright protects original works of authorship fixed in any tangible medium of expression.[13] A crucial distinction to patents is that copyright does not protect the idea behind a work of authorship, but merely the expression.[14] For example, on a drawing describing a machine, the drawing on the paper can be protected by Copyright, while the method itself cannot. Thus, while the author has exclusive rights to distribute the drawing, anyone can use the machine it describes, as long as it is not protected by patents.

Copyright extends for 70 years beyond the death of the author. During this time, the author has the exclusive right to reproduce, prepare derivative work, distribute copies, to display and to perform the work.[15]

The source code (see above) of computer programs is protected by copyright.[16] However, the underlying idea is not – this means anyone can take the algorithm, reimplement it in their own code and sell the product.

In this thesis I am going to focus on the protection of the algorithm and idea behind the software, which as mentioned above can only be achieved by patenting the software. I will not examine the protection of the software as text through copyright.

[13] 17 U.S.C. § 102(a).
[14] 17 U.S.C. § 102(b).
[15] 17 U.S.C. §106.
[16] Phillips, John C. *Sui Generis Intellectual Property Protection for Computer Software*. George Washington Law Review 60.4 (1991-1992): 1010.

3 Software patents in the United States

To see how software can be treated in the patent system, I will analyse the development of software patents in the United States, one of the largest economies and producers of software in the world. I will also explain the reasons brought up in court.

3.1 Law

In the U.S., patents are regulated in the U.S. Patent Act -- 35 U.S.C. § 101 of this law states:

> *Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.*

Article 102 covers the novelty requirement while Article 103 regulates the need for the invention to be non-obvious.

As the United States is a common law system, decisions of the higher courts are binding and therefore have an important role in establishing the meaning of the law. Thus, in order to fully understand the possibility to patent software in the United States, case law by the Supreme Court and the Federal Circuit has to be examined. As you will see, most of this focuses on whether software can be seen as a process or machine, as required by § 101. Together, the four possibilities of patenting (process, machine, manufacture or composition of matter) are referred to as statutory matter.

3.2 Case law

3.2.1 Benson (1972)

The first case that covers software patents is Gottschalk v. Benson in 1972.[17] A company had applied for a patent on a method of converting numbers from a decimal number to a binary number. This is important as computers use the binary system internally while humans use the decimal system. The patent applicant classified the claim as a process consisting of the algorithm and a computer.

[17] *Gottschalk v. Benson*, 409 U.S. 63 (1972).

The court first reiterated previous case law that a scientific truth or mathematical expression is not patentable. In this case, the court found the algorithm to be a sort of scientific truth, or an abstract idea. Only an *implementation* of such an idea can be patentable. In this case, the patent claimed the use of the algorithm on any computer. In effect, it would be a patent on the abstract idea itself. It would wholly pre-empt the use of this algorithm.[18] Therefore, the court declined the patent. It also offered some guidance on which implementation of the algorithm could have been patentable, namely the algorithm together with a specific machine or the algorithm together with a process that transforms matter.[19] The computer was not seen as a specific machine, as it was too broad and applied to all computers.

3.2.2 Flook (1978)

In Parker v. Flook, the patent was related to conversion of oil.[20] During this process, it is important that the temperature does not rise over a certain limit. The applicant wanted a patent on a new, better formula for calculating this limit. He did not describe how to collect the values to make the calculation or what to do with the limit once it was calculated. [21]

The court rejected the patent. It first found that the use of the formula to update an alarm limit did not make the formula patentable. This was due to the fact that it would otherwise be possible to patent all formulas by merely adding a small, insignificant step of what to do with the formula to the patent application.[22]

The court did not stop here. It "pretended" that the formula was part of the prior art. It then looked to the rest of the patent to see if the *application* of the formula was inventive. As all other elements of the patent were well known in the industry, it decided that so was not the case. Therefore, it rejected the patent.[23]

3.2.3 Diehr (1981)

Another important case is Diamond v. Diehr in 1981.[24] In this case, a company tried to patent a method for making raw rubber into useable rubber by applying heat and adding chemicals. The claim included a well-known formula used to calculate when the process was done on a computer.[25]

---

[18] *Id*. at 71-72.
[19] *Id*. at 70.
[20] *Parker v. Flook*, 437 U.S. 584 (1978).
[21] *Id*. at 590.
[22] *Id*.
[23] *Id*. at 594.
[24] *Diamond v. Diehr*, 450 U.S. 175 (1981).
[25] *Id*. at. 177-179.

The court again mentioned that patenting a principle is not possible. However, in this case, the software was only part of a process that performs a function protected by the patent system. The claim was not for a patent on the formula, but a patent on a process *implementing* the formula. The process did not stop being paten eligible due to the inclusion of software, therefore the patent was granted.[26]

3.2.4 Freeman-Walter-Abele

In the early 1980's, a test known as Freeman-Walter-Abele was developed by the Court of Customs and Patent Appeals and used in several cases in order to simplify the evaluation of whether a claim was a claim on a principle or on a process. It continued the Supreme Court's line of being quite strict on patents on software. The test was troublesome to use in practice, however, and has been largely overplayed in more recent rulings.[27]

3.2.5 State Street Bank (1998)

The right to software patents was expanded with the State Street Bank ruling.[28] Here, the applicant tried to patent a method to calculate the value of different connected funds. The United States Court of Appeal for the Federal Circuit decided that the method was in fact patentable, due to it producing a "useful, concrete and tangible result". It was not simply a principle or abstract idea but a practical application of such an idea for calculating a share price.[29] This opened the door for patents not just of software but also of business methods, which was previously not thought to be possible.

As a result of the new, more tolerant approach, the USPTO published a set of internal guidelines to determine whether a claim including software is patent eligible. While these were not legally binding, they were followed by the examiners and the patent lawyers. The guidelines specified a flowchart of sorts to determine if the software invention fits into one of the four statutory subject matter categories. This cleared up some confusion regarding the possibility of patenting software.[30] It also opened the floodgates for software patents, that were now flowing in by the thousands.

---

[26] *Id*. at. 191-192.
[27] Loretta Smith, *Balancing Copyright and Patent Protection for Software in the U.S. with Considerations for Marketing it in Europe*, (1998) at 51-52.
[28] *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*, 149 F.3d 1368 (1998).
[29] *Id*.
[30] Smith, *supra*, at 67-70.

3.2.6 Bilski v. Kappos (2010)

In Bilski v. Kappos, an inventor tried to patent a method for hedging risks.[31] The court began by deciding that the "useful, concrete and practical result"-test from State Street Bank was inadequate to determine whether a software was patent eligible. Instead, it went back to Benson, Flook and Diehr, and reiterated the machine-or-transformation test. In this case the claim neither was coupled to a particular machine nor transformed an article. Therefore it was rejected.[32] The judgement did not decide the machine-or-transformation test to be the only test of the patent, but more of a clue to patentability.[33]

Bilski signalled the start of a new, more restrictive view on software patenting, but failed to clarify the limitations.[34]

3.2.7 Mayo (2012)

In Mayo, an inventor had patented a method for doctors to determine the effectiveness of a medicine by measuring metabolic effects in the patient.[35] The court mentioned that "[L]aws of nature, natural phenomena, and abstract ideas" are not eligible for patenting. It then went on to determine whether the current method was part of one of these categories. The court argued that it was – the correlation between the metabolic levels in the patient and the effectiveness of the medicine was classified as a law of nature. The other, subsequent steps were all well-understood and routine and thus not enough to make the law of nature patentable.[36]

The decision made it possible to patent laws of nature only if the implementation is unconventional. The court revived the method used in Flook. It was unclear whether this was applicable to software, but this was clarified with Alice.

[31] *Bilski v. Kappos*, 561 U. S. 593 (2010).
[32] *Id*. at 16.
[33] *Id*. at 7-8.
[34] Asa Hellstadius, *Software Patents*, 56 Scandinavian Stud. L. 361, 396 (2010) at 395.
[35] *Mayo Collaborative Services v. Prometheus Laboratories, Inc.*, 566 U.S. ___ (2012).
[36] *Id*. at 10-11.

3.2.8 Alice (2014)

The first time the supreme court used the new method on software was in Alice Corp vs CLS Bank International.[37] Alice Corp held several patents related to mitigating the risk that only one contract party will uphold its contractual obligations. The claims covered a scheme with a computer system. The court used the method from Mayo of first determining whether the patent claims an abstract idea, and in that case see if the patent contains a concept inventive enough to make the patent eligible anyway. For the first step, the court decided that the method for risk hedging is an abstract idea. However, it did not further clarify why.[38]

For the second step, the court had to decide whether the steps surrounding the process were enough to make an abstract idea patent eligible. To do this, there would have to be additional, inventive features. In this case, the only additional feature was a computer. The court considered this not inventive enough and rejected the patent.[39]

Alice has fundamentally altered the possibility to patent software, and as a result many software patents have been found invalid. Examples of these cases are Digitech[40] and buySAFE[41].

3.3 Summary

Patent eligibility for software in the United States seems to have gone through several phases. The first three cases, Benson, Flook and Diehr, opened the door for software patenting. However, they placed strong limits on what was patentable, largely requiring either a specific machine or a transformation of matter. The Freeman-Walter-Abele cases attempted to clarify these rules but failed in creating a coherent framework, and were soon abolished. State Street Bank made patenting of software possible to a large degree if it could produce a useful, concrete and tangible result.

This spurred the creation of new, more tolerant guidelines and massively increased the issuance of software patents over the next ten years. Recently, however, the possibility of patenting software has decreased through the cases Bilski, Mayo and Alice. Bilski reintroduced the machine-or-transformation test from Benson. Mayo used the test from Flook of treating the algorithm as part of the prior art and examining the implementation of that algorithm to find the required inventiveness. Alice further clarified this,

[37] *Alice Corp. v. CLS Bank International*, 573 U.S. __, (2014).
[38] *Id*. at 10.
[39] *Id*. at 11-14.
[40] *Digitech Image Technologies, LLC v. Electronics for Imaging, Inc.*, 758 F. 3d 1344 (2014).
[41] *buySAFE, Inc. v. Google, Inc.*, 765 F.3d 1350 (2014).

by saying that merely claiming a method with a computer does not make software patent eligible.

The prevalent method today for determining patent eligibility today then seems to be the one established in Mayo and clarified in Alice. First, if the invention contains “[L]aws of nature, natural phenomena, and abstract ideas”, these are counted towards the prior art, and the implementation of that concept has to be inventive in order for the idea to be patentable. Merely limiting the application of the idea to a certain field in the claim does not make the idea patent eligible (Flook, Mayo) and neither does claiming a process together with a computer.

4 Debate

Software patents are a hotly debated topic. There is a large movement on the internet that is vehemently opposed to software patents. Also in academia the issue is frequently discussed, on both sides of the argument.

4.1 Software patents

4.1.1 Against software patents

One opinion against software patents is that by Richard Stallman, a prominent software developer and creator of the League for Programming Freedom. Stallman's first argument was that software patents are often granted wrongly. This was due to the Patent Office having difficulty recruiting computer science graduates. Many examiners do not have the possibility of accurately assessing whether a piece of software is widely known or obvious. This is even more difficult due to many software techniques not being mentioned in literature, as they appear too obvious or insufficiently general. Stallman mentioned several patents that he believed should not have been granted, but are very hard to invalidate due to the difficulty of finding and proving prior art.[42]

Stallman further argued that even a correctly applied patent system would be unsuitable for software. He believed that the applied standard for the inventive step is too low for the software field, where programmers are trained and used to applying the same solution to different domains, which would count as non-obvious according to the courts.[43]

The next argument relates to the risk of dealing with computer patents. A piece of software that costs 100k dollars to develop might contain hundreds of inventions. A patent search to determine that the inventions are not patent protected would cost millions of dollars to perform, and even then not guarantee that the code is not infringing on any patent. This means the creator runs the risk of being sued for unknowingly using a patented idea. This would lead to people choosing not to create a piece of software for fear of litigation.[44]

Stallman also feared a situation where large companies would obtain thousands of patents. In order to not lag behind and risk litigation, the other large companies would do the same. Then they would engage in cross-

[42] *Against Software Patents: The League for Programming Freedom*, 14 Hastings Comm. & Ent. L.J. 297, 314 (1991-1992) at 300-302.
[43] *Id*. at 302.
[44] *Id*. at 307-309.

licensing deals to allow each other the use of the protected intellectual property. However, this would shut smaller companies, that do not have the possibility of obtaining a wealth of patents out of the market. There is also the risk of "patent trolls", companies obtaining patents and making it their business to sue other companies, making software development unprofitable due to high licensing cost and the constant risk of litigation.[45]

Stallman also argued that independent reinvention of software is so common that the purpose of patents, to disseminate the newest research to the public, is not fulfilled. Everyone has access to the newest ideas anyway.[46]

According to Stallman, all of this would impede software innovation. Therefore, he suggested a method to delimit software from hardware by defining software as "built from ideal infallible mathematical components", and "easily and cheaply" copiable. He wants to separate software from hardware entirely, and only allow the hardware to be patentable, and to see the software as "an extension of the programmer's mind". Software could never infringe on patents or be the subject of patents.[47]

4.1.2 In favour of software patents

On the other hand, Jeffrey S. Goodman argued for a patent on software. He believed that the Supreme Court had misunderstood algorithms as something that is merely solving mathematical problems, while they are used to solve many different problems. He argues for a nuanced view where algorithms that lie very closely to a natural law are not eligible for patenting, while software that directs a computer to perform a new and useful process should be patentable. He also argues that software should fall within the meaning of process due to the interaction with computer hardware.[48]

Goodman believes that the granting of software patents would accelerate the software industry and the pace of technology dissemination by motivating inventors to explain their innovations. It would also make inventors more prone to risky investments and prevent wasteful duplication of research efforts. With the resulting growth of the software industry, standard of living in the U.S. would increase.[49]

He also lists several costs of granting patents on software. These are the monopolisation of software, which he believed would be solved by market mechanisms, the misuse of patents and the increased administrative costs of granting software patents.[50]

---

[45] *Id*. at 308.
[46] *Id*. at 309.
[47] *Id*. at 311-312.
[48] Jeffrey S. Goodman, *Policy Implications of Granting Patent Protection to Computer Software: An Economic Analysis, The* , 37 Vand. L. Rev. 147, 182 (1984) at 172-175.
[49] *Id*. at 175-176.
[50] *Id*. at 176.

4.2 Sui generis protection

4.2.1 In favour of sui generis protection

Steven Toeniskoetter argued for the creation of a separate, sui generis approach to software protection. He believed both patents and copyright to be inappropriate for software, and instead suggested a separate system similar to that of database protection. This new system would require novelty and an inventive step, but no requirement for industrial applicability. The protection term would also be significantly shorter, two to five years, and there would furthermore be a compulsory license provision after reasonable negotiation, in order to not impede the progress of technology.[51]

4.2.2 Against sui generis protection

John M. Griem argued against a sui generis approach, due to software being sufficiently protected through a combination of copyright and patents, and the flaws of an incomplete database of prior art and unqualified examiners being solvable by other methods. He strongly disagreed with the Benson arguments. His argument is that formulas are not the same as algorithms, as they ignore the human effort and the innovation that went into creating the algorithm.[52] He believed most of the problems applying the novelty requirement and inventive step rose from the misguided Benson decision, causing the PTO not to invest in the facilities needed to perform the analyses.[53]

[51] Steven B. Toeniskoetter, *Protection of Software Intellectual Property in Europe: An Alternative Sui Generis Approach*, 10 Intell. Prop. L. Bull. 65, 82 (2005-2006) at 76-79.
[52] John M. Jr. Griem, *Against a Sui Generis System of Intellectual Property for Computer Software*, 22 Hofstra L. Rev. 145, 176 (1993-1994), at 164.
[53] *Id*. at 172-174.

5 Analysis

We have now seen what the courts in the U.S. think about software patents and their rationalisations. We have also examined voices from the legal doctrine, some of whom were opposed to software patents while some were in favour. I will now analyse the material from a philosophical and economic perspective and arrive at the result of how to properly protect software using intellectual property.

5.1 Legal characterization of software

At the deepest level, the question is a philosophical one – is software discovered or invented? Most people agree that for example the relation between a circles radius and its circumference, $C = 2\pi r$, is simply a law of nature. It is inherent in the world, and simply deduced by humans. What this equation does is to act on one set of numbers and produce another set of numbers. As computers contain memory that is comprised of bits (1 or 0), everything a computer does is taking one set of numbers and turning them into another set. The first set of numbers can come from bits on a CD, keys pressed by the user, an image recorded from a camera or a server on the internet, while the second resulting set can be an output on a screen, a sound or a request to a server somewhere on the internet. This also means that given enough time, a human could read the machine code and perform the calculations on a (huge) piece of paper, and arrive at the exact same result as the computer.

What then is the difference between the simple formula for calculating the circumference of the circle and the complex algorithms that for example take a video stream from the internet and display it on our phone? In Benson, in essence, the court claimed there was none – the algorithm Benson tried to patent was seen as a law of nature, and the patent was denied. The algorithm was seen as a basic building block of science, just like the pi-formula. Benson however opened the door for the patenting of processes including algorithms, if they passed the machine-or-transformation test. If the algorithm thus is part in a process that transforms matter or bound to a particular machine, it is more than just repetition of a law of nature, and therefore patent eligible.

In my view, this misses the essence of software. As Griehm argues, reducing all software to a formula is oversimplifying the matter. While some software is indubitably only a direct application of natural laws, other software is much more than that. A program for calculating the circumference of a circle from its radius, for example, is merely the application of a natural law. If a programmer however implements an operating system such as Windows, it is the implementation of millions of man-hours of hard work. It also solves several technical problems, such as how to distribute resources between programs and precisely which signals

to send to the screen. Calling this merely a direct application of laws of nature misses the creative and system building the creator of software undertakes.

The point of patents is to protect the implementation of an idea. It has been purposefully built in a way to be agnostic of technology and field. After passing the basic requirements, any invention can be protected. It should not matter whether the invention comes in the form of hardware or software. Software today often solves the same problems that hardware does. Why should one be patentable when the other is not?

That does not necessarily mean that the Benson patents should have been granted. In order for software to be patent eligible, it would still have to fulfil the other requirements of patents, including the requirement of utility, novelty and the need for an inventive step. The algorithm presented in Benson seems to miss the inventive step, as it is an easy recalculation.

In Flook, the inventor had used a formula to calculate an alarm limit used in petroleum processing. The court found this to be insufficient to make the invention patent eligible, as it simply limited the scope of the application and added rudimentary post-solution steps. I disagree with the reasoning of the court. That the claim consisted merely of software should not exclude the invention from patentability, as stated above. If the inventive step requirement and novelty requirement was fulfilled, the patent should therefore have been granted. A new formula for calculating a value used in an industrial process is an invention just as another part that improves the efficiency of the process would be.

In Diehr, a patent on a process involving an algorithm was granted. This was due to the algorithm in this case being integrated in a system that was itself patentable. Thus, the algorithm itself was not patented but the system the algorithm was integrated in. The court decided that the entire process was patentable and that the inclusion of an algorithm did not change this. I believe this conclusion to be correct. The patent in this case was not granted for the algorithm at all, but to the process as a whole. Therefore, the court did not have to assess the patent eligibility of software. As the implementation in this case is inventive, the patent was granted, as it should have been.

In State Street Bank, the court totally changed the eligibility test and instead focused on the software's possibility to produce a "concrete, useful and tangible result", which opened the door for a lot of software patenting. I believe this was a reasonable approach. The utility requirement of patents is already part of the patent law, and is a requirement for all patents, including software. As long as the other requirements were fulfilled correctly, the granting of this patent was justified.

In its more recent rulings, the supreme court has come back to software patentability and issued a few landmark cases. These have established the

Alice method of treating software as prior art, and judging the inventiveness of the implementation in order to determine patentability. As stated above, I believe this to be a too narrow requirement for patent eligibility. If the other requirements for patent eligibility are fulfilled, there should be no barrier to patenting software.

5.2 Practical considerations

From a legal characterization point of view, I have argued for the inclusion of software by itself in patents. But law, and especially patent law, is at its basis economical law. Does it make economic sense to make software patent eligible? Patents give inventors a large monopoly on the idea behind an invention. In order for this to be justified, the purpose of the patent has to be fulfilled. Is it in the case of software? Or is there something fundamentally different about software, making patent protection unjustified?

The first purpose of patents is to motivate the market to innovate. Stallman suggested that the current patent model has a negative effect on innovation in the market by creating a legal minefield for software developers. Goodman believed software patents would increase innovation by increasing investment in research and development.

Many of the arguments of the opponents of software patents relate not to the inclusion of software patents in the itself, but to the application of the rules around software patents by the US patent office. Stallman criticized the patent office on mainly two points, the overlooking of prior art and the granting of obvious patents. Neither of these are problems inherent to patenting software. The solution to patents being granted despite prior art is to train examiners properly and to establish a database of prior art for software. While Stallman argues that this is impossible for software, someone obviously does have to know of the existence of prior art – otherwise it would not exist. Examiners that are trained programmers or a system for asking experts in their opinion could go a long way to solving these issues.

The same goes for the implementation of the inventive step. Patents on inventions that are obvious, and would be reinvented by other programmers facing the same issue, should not be granted. This would decrease the number of patents and create a situation where only truly unique inventions are protectable, as it should be. Many of the patens the courts have rejected, such as Benson, should have been rejected due to not being inventive enough rather than due to being excluded from patentability. As less patents are granted, the expense of a patent and the risk of being sued for unknowingly having recreated a patent protected invention would decrease.

I would like to highlight that the solution of these problems is crucial, without them innovation is significantly impeded and incentives for parties

on the market are misaligned. However, I see no significant difference between software and traditional inventions, therefore patent protection is well suited to deal with software when the problems are solved.

5.3 A shorter protection term?

However, while the patent system does seem suited to deal with software, there is much that seems to point in the direction of the decreasing the protection term, as argued by Toeniskoetter.

The current term of 20 years has been more or less consistent since the 19th century. It was agreed upon in the light of hardware development. In fields such as medicine and utilities, the time from invention to implementation is much longer. After the invention has been made, the company still needs to invest considerable resources into making the invention into a product. This can take many years, which is reflected in the long protection term. After the product hits the market, the company still needs several years in order to recoup the investment. The situation is different in the software industry however. Here, the invention is often made in the process of development, and already implemented when the patent is claimed. Thus, the selling of the invention can start immediately, and a term of 20 years is not required to recoup the investment. A shorter term, such as 3 to 5 years, would be completely adequate.

The software industry is also considerably more volatile and innovative. In an industry where many of the biggest players, such as Google and Facebook, are less than 20 years old it seems excessive to grant a patent that would monopolize an idea for such a long time. The inability of competitors to use the invention and further innovate that area comes at a massive cost to the economy. Thus, a shorter term would be more appropriate.

A shorter term could also reverse the trend of patent trolls applying for basic patents and then using them for their own gains, impeding innovation in the process. By granting only short terms, this would significantly reduce the profitability of these patent trolls. However, by applying the inventive step correctly, much of this issue should be solved anyway as only truly innovative inventions that deserve protection are premiered.

5.4 Summary

In conclusion, I believe that software should not be excluded from patentability. Patents are meant to protect inventions in all fields. Software inventions can be just as innovative and important as hardware inventions, therefore I do not see the need to distinguish between the two. The practical

problems of software patents seem to originate from a misapplication of the novelty and inventive step requirements. These issued can and should be solved. I do believe we should consider a shorter protection term of software, however, due to the pace of the industry. This could be done by creating a new form of protection similar to patents or by altering patent law.

Bibliography

Sources

Commission of the European Communities, Proposal for a Directive of the European Parliament and of the Council on the patentability of computer-implemented inventions, COM (2002) 92 final, Explanatory memorandum

WIPO Model Provisions on the protection of computer software, Geneva 1978

Literature

Against Software Patents: The League for Programming Freedom, 14 Hastings Comm. & Ent. L.J. 297, 314 (1991-1992)

Goodman, Jeffrey S., *Policy Implications of Granting Patent Protection to Computer Software: An Economic Analysis, The* , 37 Vand. L. Rev. 147, 182 (1984)

Griem, John M. Jr., *Against a Sui Generis System of Intellectual Property for Computer Software*, 22 Hofstra L. Rev. 145, 176 (1993-1994)

Hellstadius, Åsa, *Software Patents*, 56 Scandinavian Stud. L. 361, 396 (2010)

Maunsbach, Ulf & Ulrika Wennersten, *Grundläggande immaterialrätt* (2 ed. 2011)

Phillips, John C., *Sui Generis Intellectual Property Protection for Computer Software*, George Washington Law Review 60.4 (1991-1992)

Smith, Loretta, *Balancing Copyright and Patent Protection for Software in the U.S. with Considerations for Marketing it in Europe*, (1998)

Toeniskoetter, Steven B., *Protection of Software Intellectual Property in Europe: An Alternative Sui Generis Approach*, 10 Intell. Prop. L. Bull. 65, 82 (2005-2006)

Table of Cases