Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Open Source + Software Patents = Innovation?

Understanding software patent policy's effects on open source innovation

Marcus M. Dapp, ETH Zurich[†]

ECPR Conference "Frontiers of Regulation", September 2006, Bath (UK)

Comments very welcome – Please do not quote

Table of Contents

[†] Center for Comparative and International Studies, ETH Zurich, Switzerland. See: www.ib.ethz.ch/people/mdapp.
This research is supported by ETH Research Grant "TH -2/05-2".

1 Motivation

Free Libre and Open Source Software, or FLOSS, has seen wide-spread adoption over the last decade. An increasing number of companies[1], governments[2], and public institutions around the world are switching to GNU/Linux[3], a prominent free operating system. While it has shown high rates of development (and deployment), there is controversy among policy makers, companies, and the free and open source community whether patenting poses a specific threat to its further growth or not. Although free and open source software is a research topic, the particular effects of patenting on its innovation have not yet been investigated systematically. I offer a first attempt.

The debate about the *patentability of software* is highly divided and politically burdened. The EU directive on the "Patentability of Computer-Implemented Inventions" was disputed heavily until its unexpected final rejection by the European Parliament in June 2005. The rejection means diverging policies of two large economic powers, Europe and the United States, in a key industry[4] of the 21st century.

FLOSS development works quite differently from the proprietary model the software industry is based on. It represents an unusual property regime by giving users the right to run, examine, modify, and redistribute the software without constraints. FLOSS represents an unusual innovation model, too. It is developed in a strictly decentralized manner, not embodied in firm-like structures for resource allocation but self-managed in international projects and largely without monetary incentives. The established conception of innovation, in which firms create and their innovations are necessarily commercialized, is put into question.

Considering software as patentable subject matter is receiving increased scrutiny and investigation from economic (Benkler 2002, FTC 2003), legal (Hilty and Geiger 2005, Lessig 2002), and policy (Kahin 2001, Shadlen, et al. 2005) perspectives. FLOSS has increasingly been picked up as a topic of scholarly research in economics (Lerner and Tirole 2001, von Hippel and von Krogh 2003), law (Lessig 2002, Boyle 1996), political science (Weber 2004), and sociology (Grassmuck 2004). Despite their links, both bodies

1 Although the European Information Technology Observatory acknowledges that the speed of the "fastest-evolving technology in history", the Internet, is mainly due to the open source approach, they provide no market figures (EITO 2004, 128). David A. Wheeler, however, maintains a collection of statistics titled "Why OSS/FS? Look at the Numbers!" at www.dwheeler.com (04/27/2006).

2 The press release for the UNCTAD 2003 E-Commerce and Development report highlights that FLOSS has "major implications for developing countries, reducing barriers to market entry, cutting costs and facilitating the rapid expansion of skills and technology". See also http://unctad.org/ecommerce (05/18/2006).

3 Wikipedia's article "Linux adoption" provides an overview (04/27/2006).

4 Europe and the U.S. account for more than half of the worldwide ICT market value, which is expected to exceed € 2,000 billion in 2006. Within ICT, software is the strongest growing market segment globally (EITO 2006, 190p).

of research have not been systematically connected yet.[5] This paper aims to contribute to the debate by offering a theoretical argument that explains the effects of software patenting on FLOSS innovation, as to date, "very little systematic analysis has examined the implications of patents for open source" (Lerner and Tirole 2004, 27).

This interdisciplinary field is still young and characterized by complexity at the intersections of software engineering, intellectual property policy and innovation economics, difficulties in defining key terms, and a still small body of literature. This paper focuses on FLOSS because although I believe FLOSS has reached size and significance which merits dedicated specific research, I think that software patent research so far has addressed FLOSS insufficiently; either only as a side aspect or not as a real special case (which I think it is).

Chapter 2 reviews what we know about patents and innovation in (non-software) industry. After explaining more in detail what software is, chapter 3 describes the general process of how software is created and what characteristics make software a special case of innovation. Chapter 4 extends the explanation to the specific characteristics of the open source model. I focus specifically on the motivations of the contributing developers. After a brief general introduction to the legal protection of software, chapter 5 presents arguments as to why software patents may or may not be of advantage to proprietary software innovation.

In chapter 6, I offer a new line of argumentation about the effects of patenting on FLOSS innovation by relating the specific motivational setup of FLOSS developers (section 4.2) to the general argumentation about software patents (chapter 5). I argue for two possible effects: FLOSS developers are either indifferent to the potential advantages of patenting because the incentives provided by patents do not address the diverse set of their motivations. At the same time, software patents may harm FLOSS innovation, as developers have to cope with the potential disadvantages of patents. It is an empirical question, whether these arguments hold or not and which one is dominating.

If this argument holds, the one-size-fits-all approach of patent law may have to be reconsidered for FLOSS in particular (and perhaps software in general). The effects of the patent regime need to be understood better if innovation policy is to support FLOSS as a field of innovation of growing importance.

5 With the exception of Lutterbeck's expert opinion for the German Federal Ministry of Economics and Technology about the effects of software patenting on security. Here, he formulated several recommendations for patent and open source policy (Lutterbeck 2000).

2 Industrial innovation and patents

> *"If we did not have a patent system, it would be irresponsible, on the basis of our present knowledge of its economic consequences, to recommend instituting one. But since we have had a patent system for a long time, it would be irresponsible, based on our present knowledge, to recommend abolishing it."*
> *Machlup 1958*

Two aspects are prime in shaping economic performance: efficient production and effective technical progress, i.e., developing knowledge for new products or production methods (Viscusi, et al. 2001). The public-good character of knowledge, however, leads to an incentive problem. Making new knowledge available to the public would ensure its optimal use because it is non-rival. But, as it is also non-excludable, we expect the creators' incentive to vanish – unless he is rewarded in some way (Fisher 2005). One possible solution is "intellectual property rights". With them, however, a fundamental trade-off occurs. To the extent innovation is encouraged through property rights, there is an underutilization of knowledge (Arrow 1962).

A *patent* is a set of exclusive monopoly rights granted by the state to an inventor for a limited amount of time. It provides the right to prevent others from making, using, selling, offering for sale, or importing the patented invention. An idea is patentable if it is an invention, which is new, has an non-obvious inventive step, and is suitable for an industrial application. When the patent is granted, all other implementations of this idea require permission of the patent-holder. In return, the patent-holder is obliged to disclose the invention to the public so that somebody skilled may replicate it.

The debate about patents' effects on innovation is not new. Studying the big patent controversy in the mid-19th century, Machlup and Penrose (Machlup and Penrose 1950) give four classical justifications for patents used in the patent controversy, which are also discussed by van Dijk 1994, Fisher 2005, and Heinemann 2002.[6]

Reward theory states that an inventor deserves compensation and fair reward, proportional to the usefulness of the invention to society. The counter argument says that patents are still unnecessary because other kinds of rewards follow as well without them (van Dijk 1994). Would patents be sufficient as incentive?

Incentive theory is independent of questions of justice and therefore "probably the most quoted argument in favour of patents" (Dutton 1984, 20). It gives a utilitarian, economic justification for patents. They enhance innovation because they give an incentive to the individual to invent and commercialize the invention by preventing imitation. Patents directly increase the profit of the inventor and discourage competitors

6 I skip *Natural right theory,* which gives a moral justification based on Lockean labor theory: "The labour of his body, and the work of his hands (...) are properly his." (Locke 1690, Sect. 27; see also Drahos 1996, 43). The counter argument is that this justification does not solve the problem of knowledge being non-rival.

to create profit from free-riding. As useful inventions increase social wealth, inventing should be encouraged through patents as they are a inexpensive means of providing these incentives (Merges, et al. 1997; Machlup 1958).

Incentive theory relies on four assumptions about the economics of the inventive process (Fisher 2005; Machlup and Penrose 1950). Uncontested are the assumptions that growth and industrial progress are socially desirable and that invention is necessary for this progress.[7] The assumption, however, that external incentives are required to induce invention – "the idea that there [is] a need to stimulate invention" (Fisher 2005, 14) – is contestable. An invention's inherent utility for the inventor can be enough to stimulate inventive activity.[8] A patent monopoly, however, locks out competitors and diverts funds from research and development into legal costs. Innovation may be reduced.

Finally, the assumption that patents are the cheapest and most efficient way to provide incentives is directly challenged by arguing that the patent system requires an expensive bureaucracy prone to inefficiency and indirectly by proposing alternative incentive systems like rewards.

Exchange theory argues that patents offer a fair balance between the public's and the inventor's interests. They encourage inventing and making the invention publicly available by requiring disclosure. Two counter-arguments can be made. First, inventions can happen independently, making disclosure likely to happen because keeping an invention a shared secret without knowing "the others" is difficult. Second, exploiting an invention without disclosing it is hardly possible (van Dijk 1994).

Only the economic theories have "survived" until today: incentive, exchange, and reward theory (Fisher 2005). Mazzoleni and Nelson offer two more arguments (Mazzoleni and Nelson 1998). To reward theory: patents induce the needed investments to develop and commercialize them. For example, a patent can help to raise investment funds or to distribute and coordinate development efforts among several companies. To incentive theory: a patent facilitates the exploration of broad prospects, allowing the patent-holder to exploit all directions. This effect largely depends on whether different inventors would follow similar paths of research or not.

Looking at the body of empirical studies, the efficacy of patents in fostering innovations are inconclusive. The 1988 patent law reform in Japan introduced multiple-claim patents for all patents and a longer patent protection in the pharmaceutical industry. Econometric analysis of Japanese and U.S. patent data of 307 Japanese

7 An empirical look at the social value of innovation in the nineteenth century can be found in Menell 2000, 134.

8 The president of the U.K.'s Institute of Civil Engineering (1851): "I think people will always invent anything that is useful and good, if it will answer their purpose to do so, even without reference to a patent." (Fisher 2005, 15). I will come back to that argument when discussing developers' motivations in chapter 4.

companies showed that neither R&D spending nor innovative output increase could be attributed to the reform (Sakakibara and Branstetter 2004). The results challenge the notion that broader patents will induce additional innovation, according to the authors.

Based on a U.S. survey of 1478 R&D labs in the manufacturing sector, Cohen et al. found that firms protect profits from invention with a range of mechanisms. Of these mechanisms, secrecy and lead time are emphasized by the majority of manufacturing industries, whereas patents are emphasized least. Patents are most effective in protecting drugs and chemicals (Cohen, et al. 2000).[9]

Arora et al. reach similar results. They used survey data for the U.S. manufacturing sector to estimate the increment of the value of an innovation realized by patenting it, and then analyzed the effect on R&D of changing that premium. Their findings imply that patents stimulate R&D across almost all manufacturing industries. The magnitude of that effect varies substantially and is highest in drugs and biotechnology. On average, patent protection provides a positive premium in only a few industries: drugs, biotechnology, and medical instruments (Arora, et al. 2003).

Sattler analyzed the perceived effectiveness of patents for protecting new products in comparison to other means of appropriation. A comparison of five studies[10] from the U.S., EU countries, and Japan showed that competitive instruments like lead time and superior sales/service efforts are ranked as the most effective means, whereas protection through rights like patents are perceived to be least effective (Sattler 2003).

Levin's U.S. study surveyed 650 R&D executives in 130 different industries about the effectiveness of seven instruments in "capturing and protecting the competitive advantages of new and improved production processes".[11] Patents were viewed as an effective instrument for protecting new technology in most chemical industries (including the drug industry), but to be relatively ineffective in most other industries (Levin, et al. 1987).

In summary, the empirical picture, whether patenting leads to more innovation or not, is inconclusive. In most industries, patents do not play a significant role in fostering innovation. Other appropriation mechanisms are deployed instead.[12] Looking at software and its specific characteristics, the picture blurs further.

---

9 Growing use of secrecy – the protection mechanism of today's software industry – was the biggest change noted.

10 The five studies are: Levin, et al. 1987; Arundel 2001; Wyatt, et al. 1985; Harabi 1995; and König and Licht 1995.

11 The seven instruments are: patents to prevent duplication, patents to secure royalty income, secrecy, lead time, moving quickly down the learning curve, and sales and service efforts.

12 See Bessen and Hunt 2004a, Heinemann 2002, and Mazzoleni and Nelson 1998 who came to similar conclusions.

3 Software as an innovative process

Information is information, not matter or energy.
Wiener 1961

Several particular characteristics give software its distinct nature and provide the basis for the economic analysis in this chapter. As these characteristics also shape the way software is created, outlining the basic process of engineering a program from the perspective of an individual developer will help us understand why "innovation" is a concept hard to maintain in the software context. I close this chapter by describing the established innovation model of the software industry, which partly leverages on software's particularities and partly effectively reverses them.

Software consists of encoded information (program instructions or data), as opposed to the physical hardware which is used to store and process software. *Program* refers to executable software as a collection of source of object code. *Source code* is a series of program instructions written by a developer in a human-readable programming language. The source code is then converted into a semantically equivalent form, *object code*, which is executed ("run") on the computer. I will use the terms software and program interchangeably, always referring to instructions.[13]

3.1 The quintessential digital good

According to Quah, a *digital good* is an economically valuable string of binary digits, which one can copy identically, modify easily, and distribute instantly across digital networks. The fact that no "matter" needs to be moved for these activities allows these operations to happen at zero marginal costs.[14] That leaves only production to require significant investments.

Digitality exhibits various economic characteristics. First, software is non-rival and (per se) non-excludable, hence a *public good*. Non-rivalry reigns because the use of a program by one person does not limit the utility of it to another person; he just copies the program. Non-excludability shows for example, when a program is available online, nobody can be excluded from using it. Access and free-riding are easy and therefore countered by legal or technical protection (Quah 2002). Scotchmer gives a detailed economic analysis of why public goods should be provided for free (Scotchmer 2004, chapter 2).

13 See also Wikipedia's articles on "computer software", "computer program" and "source code" (04/28/2006).

14 Of course, physical delivery of digital goods (burning and shipping discs to customers) is costly. Economically, however, it is inefficient as digital distribution over networks would avoid physical carriers like discs. This shows that existing distribution methods are not yet fully adapted to digital goods.

Instant and lossless copying at close to zero costs leads to considerable scaling effects. First, economies of scale occur because a supplier is capable to instantly scale the production (Hoppen, 2005). While a first instance of a program is expensive to produce due to fixed costs, later digital copies are very cheap (Shapiro and Varian 1999). But "expensive" is a relative term: Compared to industries producing physical goods, investments in software are significantly lower for two reasons: a) personal computers are cheap standard devices and mostly not specialized and expensive (FTC 2003, 45); b) no "matter" is involved during the production process; everything happens digitally. Also, strong scaling effects mean that a rise in demand can be met in a short time. In other words, a producer can reach significant market share in a short time (Hoppen, 2005) and experience pronounced first mover advantages.[15] A second aspect of gratis copying are increasing returns to scale for producers. They occur when output increases more than proportionately after input was increased by some amount. In the software industry, the share of profit increases for each additional package sold.

Another characteristic which distinguishes software from other digital goods is *functional utility*. Running programs are useful because they process information faster and more precisely than human beings. Their value is derived from that problem-solving capability. Customers' willingness to pay for that capability determines the price, not the property value of the software. This gives rise to differential pricing strategies, where the same program can have very different prices, depending on who the customers are (Lutterbeck 2000, Shapiro and Varian 1999). Another aspect is that utility degrades over time because users' needs change constantly. The day a new program is out its value is highest. Unless the program is constantly adapted, utility will gradually decay and even drop to zero when a better version becomes available.

The utility argument goes even deeper: All digital goods are created through programs – including programs themselves. Software is the universal means of production, the "quintessential digital good" (Quah 2002, 29), with a unique triple role of being the blueprint, the producing machine and the final product all in one. The moment of invention and the moment of "physical manifestation" of that invention are identical. Quah calls innovation "the first instantiation of a digital good" (Quah 2002, 7).

A third characteristic that distinguishes programs from other digital goods is *interoperability*. Programs are called interoperable if they are capable to exchange data, process the same data formats or use the same protocols. Interoperability leads to strong positive network externalities (Quah 2002). The value of using a certain program

15 Low entry barriers can help explain why the European software market mostly consists of small- and medium-sized enterprises. Strong first mover advantages can help explain why a few multinational companies dominate the U.S. software market.

increases the larger the pool of users is with whom one can exchange data. Likewise, changing to another, non-interoperable program puts considerable switching costs on customers.

3.2 Engineering process and innovation

The *engineering process* for software has three steps: i) Specifying the problem in structured but natural language to describe what problem the software should solve. ii) Implementing one possible *algorithm* (series of instructions) in a suitable programming language. There is choice for the developer to outline the algorithm as well as choose the programming language. The source code finally describes how the problem is to be solved. iii) Converting the source code into executable object code with the help of a compiler program. The executable file can be run on a computer. I follow the definition of *software development* as "the new or further development, as well as adaption, of computer programs with the aid of a programming language" (Blind, et al. 2005, 5).

Using the terms "invention" or "innovation" in the software context is technically difficult. They convey a notion of software as a discrete product, where one product can be very different and hence an innovation compared to another. But comparing programs is difficult: New user functions may be perceived as new although the underlying source code is not new at all. Two programs performing the same task may do so in quite different ways. Although one may be more "innovative" than the other, a user may not recognize the difference.[16] I suggest focusing on the *developer side* when referring to innovation in software engineering. Additionally, I suggest that the term locates on a higher abstraction level than source code syntax. According to Wheeler, most of what happens on the source-code level, is recombination or integration of existing components. I support his approach in restricting the term "software innovation" to the developer side and therein to the introduction of *new programming paradigms* (Wheeler 2001).

The technical development of (large) software systems has been described using the *laws of software evolution*, pioneered by Lehmann three decades ago (Lehmann 2002). But as they "do not provide a rich or deep characterization of the evolution of F/OSS systems" (Scacchi 2003, 27), I will not consider them in this paper.[17]

Two sources of innovation lie in software. One is the competition of independent ideas. One program is only one possible implementation of a solution. There exist different, independent paths to create source code that can all lead to programs

16 Klincewicz goes a step further by raising the question whether the majority of FLOSS projects is innovative at all. He argues most are "me-too" clones and have not been innovative on their own (Klincewicz 2005).

17 Koch also suggests that some of the long-standing laws of software evolution are violated by the open source model after finding that several large open source projects are able to sustain super-linear growth(Koch 2005).

performing the same or very similar functions. This allows competition between different solutions. Yet, imitation – the opposite of independent ideas – is an equally important source as we will see later. Two software innovation models have developed over the last decades diverging over these two sources of innovation. The established one, usually called the "proprietary model", is described next.

3.3 The proprietary model

Standard proprietary software differs in two dimensions from FLOSS: the innovation model (chapter 4) and the approach to legal protection (section 5.1). Both models handle the above mentioned software characteristics in different ways. I present the proprietary model briefly to provide a starting point from familiar ground.

The first characteristic – software being a public good – is countered by two different modes of protection: copyright and trade secrets. Copyright protects the otherwise unprotected source code from identical copying, whereas trade secrets allow to keep source code secret and inaccessible to competitors and customers. Together, they turn software into a marketable private good. Firms sell boxes containing software discs and serial numbers for activation. Sharing of source code is considered undesirable and thus is legally prevented. Both legal rights to the software are a key precondition for the functioning of this model.

The second characteristic – functional utility – is leveraged in marketing. Selling software is difficult if the customer cannot see and feel the improvements. The more visible changes there are, the easier it is to sell the program. At the same time, the danger increases that users are challenged with a steep learning curve, which might harm adoption of the new version.[18] Price is justified primarily by new functions and competition is set up around functionality and comfort. Fulfilling a new customer need is often called an "innovation", whereas "real" innovations according to Wheeler's definition are rare (Wheeler 2001).

Third, interoperability can be used strategically to defend markets and bind customers. For example, firms promote internally developed and secret data formats and standards in their programs to create barriers for competitors to create alternatives. This increases switching costs for customers and ensures a steady income stream as long as customers stay with the product throughout new releases. In the extreme, this can lead to a situation of lock-in: customers depend on programs and the respective producer because everyone else uses them, too.[19] Similarly, the strategy to win new

18 Focus on new and more functions may gradually lead to "feature bloat". The increased complexity in using a program with many features may reduce overall utility.

19 The competition between Microsoft's office suite and "OpenOffice.org" illustrates this. Although the latter is considered sufficiently similar in functionality to the first, customers are hesitating to switch because the existing

customers is to drive a low-price strategy at the start. When a critical user base is acquired, network effects will set in, supported by non-interoperability strategies. Non-interoperability combined with scaling effects can make the size of the install base a critical success factor.[20]

A fourth, production-related characteristic is the *particular and short life cycle* of software. The development cycles are short because development labs, production machines or clinical tests are not required. Typical steps like conception, testing, production, etc. can be done on the same (universal) computer. Many solution paths can be simulated digitally in a cheap way, which leads to short development times, often measured in months rather than in years (Blind, et al. 2003). On the market side, we also see a "short effective life of software innovations" (Cohen and Lemley 2001, 4), because as users' software needs change quickly and adaption can be provided quickly as well, new versions of programs are released in short succession. They usually render the existing version on the market obsolete. These dynamics lead to a tippy market where success depends on speed and first mover advantages on the supply side and user loyalty on the demand side (Hoppen, 2005).

In sum, we observe that software by nature is vastly different from physical goods. This raises the question whether it should be treated differently by innovation policy when thinking about FLOSS's innovation, production, and distribution mechanisms (next chapter).

pools of users and documents is very large.

20 Internet telephony software "Skype" is an informative example. Skype uses proprietary protocols, which do not allow users to make calls to non-Skype users. Therefore, competitors, even if they are using open protocols among them, can technically not interoperate with Sykpe and are thus hampered in competition.

4 Open source innovation

"Whether or not incentives are needed, and if they are, how strong they need to be to generate innovations that would otherwise not take place, are hotly debated questions."
CED 2006

This chapter describes the key characteristics of the open source model. The first section argues that the model in itself is innovative in various respects. The particular incentive structure of the individuals driving this model will be examined in the middle section. Finally, I provide an overview of the existing theoretical models, which describe the workings of the open source innovation model.

Besides the acronym FLOSS ("free libre open source software"), I use the different terms interchangeably, albeit with some preferences: "Open source" (OSI 2006) is given preference when discussing innovation models and "free software" (FSF 2006) when discussing property relations.[21]

4.1 An innovative model

Applying Amabile's definition of innovation as the implementation of ideas through social, commercial or organizational activities, the open source model can be considered an innovation in itself(Amabile 1996). I start by describing the model's key aspects in these three dimensions before I pick up and extend the characteristics introduced in chapter 3.

The *sharing of (source) code* is the social activity at the heart of the open source model. Accordingly, the continued free access to the source code is assured – a key prerequisite for the sharing to function. Benkler argues that the personal computer as a cheap, universal means of production and the Internet as a ubiquitous and cheap means of many-to-many communication, have removed the physical barriers on information production which required market-based strategies based on exclusive rights to undertake the high investments needed. Declining costs for this infrastructure makes non-market and non-proprietary models increasingly more feasible. Communicating and collaborating unrestrictedly over the Internet supports the production of information goods through "coordinate effects of the uncoordinated actions of a wide and diverse range of individuals and organizations acting on a wide range of motivations". Open source software is an example of how such large-scale cooperative efforts can be effective (Benkler 2006, 4-5).

Giving source code away *free of charge* is the prominent commercial particularity of the open source model. Along with that goes the fact that many developers do not earn

21 For details on differences see (Klang 2004).

money contributing; for many, it is a volunteer effort. No typical price-based market forces play because traditional producer-consumer-relationships do not exist as the programs are not sold. Hence, a project's success cannot be measured in commercial terms.

While open source projects compete on the technical merits of the programs and the attraction of good developers, companies engaged in open source development continue to compete in the typical services of the software business: e.g., training, maintenance, and customization.[22] Producing software in a collaborative manner and giving it away for free is not incompatible with a software business (Goldman and Gabriel 2005; Ghosh 2005). Several companies, like IBM, HP, Novel, Sun, and Red Hat, are building new business models in line with the rules of open source development.

Organizationally, individual developers participate in a *decentralized and collaborative effort*. Virtual communities gather around "projects" hosted on openly accessible websites. The Internet allows for fast communication and coordination and effectively lowers entry barriers for new contributors. Code is written, copied, and recombined with other code, while access to all code is legally guaranteed (see 5.1).[23]

All three attributes accommodate and leverage two additional characteristics of software development not discussed so far, as they become only fully relevant in an open source environment. I frame their discussion using the evolutionary innovation model by Saviotti (Saviotti 1997; Marinova and Phillimore 2003). In evolutionary theory, innovations are seen as mutations within a concept of generation of variety.

The first additional characteristic, *code variety*, denotes the fact that different solution paths exist to solve a specific problem in software. Different pursuits of independent developers can lead to faster innovation and higher probability to achieve an innovation goal – if the solutions are openly shared. Bessen and Maskin speak of "complementary innovation" (Bessen and Maskin 2000). A key reason for this variety to occur is the heterogeneity of users' needs (Bessen 2005) and developers' interests. Different developers see different prospects to build on when confronted with a certain program.[24]

Another concept of evolutionary theory is reproduction and inheritance. Here, the second additional characteristic comes into play: software development happens in a strongly *cumulative way*.[25] Many small changes and additions build up the source code

22 For an analysis of open source business models see Krishnamurthy 2005.

23 Many large open source projects have established own foundations covering a variety of roles (O'Mahony 2005).

24 Open source projects know a measure called "forking" of projects. This may happen when no consent on the overall direction can be found and therefore increases the number of approaches to solve a certain problem.

25 While stressing different aspects, the terms "incremental" (Scotchmer 1991) and "sequential" innovation (Bessen

over many iterations, improving it by replacement or addition of functionality. Developers "copy & paste" existing code fragments to build new recombinations and adapt them to the problem at hand (Quah 2002, 29). Thus, a program may grow by thousands and millions of "lines of code" up to a level of complexity where the whole cannot be understood by a single individual anymore. Yet, despite its overall size, a reasonable cut-out chunk is still understandable to a skilled programmer, enabling him or her to contribute to the project. Consequently, even a large group working side by side can be very effective in developing software.[26]

Hoppen points out that the complementary and cumulative nature of innovations, particularly in high-tech industries, is widely recognized and cites several empirical studies in support (Hoppen, 2005). This makes it possible for the open source model to manage the four software characteristics (chapter 3.1) quite differently.

First, the positive network externalities of software as a public good can be leveraged fully because exclusive rights are not sought after. Open source is decidedly non-excludable. Different factors encourage the wide distribution of programs on the demand side: low costs, a permissive copyright regime, and the adaptability of the software to own needs. On the supply side, large and visible projects with many users attract more developers. This helps the project as a whole to grow.

Second, in the absence of market pressure, it is easier to develop software based on technical considerations rather than following market and business considerations. Not compromised by market strategy, functional utility is likely to be higher.

Third, for the same reason, developers can pursue interoperability as a technical goal without the need to balance the trade-off with a business goal to bind customers. Interoperability helps different open source programs to interact more effectively, which increases overall utility for the user. Finally, it makes code sharing easier and reduces switching costs for users.

Fourth, life cycles of a FLOSS program are even shorter than in the proprietary model. One reason is that development happens more incremental with more but smaller changes and shorter periods between releases as in the proprietary model. As no market pressure is present and no market-driven deadlines are to be met, a release is likely to be tested more thoroughly before being released.

Two questions, however, are still left open and will be tackled next: What is the motivation for developers to contribute under such a regime? And who is participating?

---

and Maskin 2000) are based on the same idea.

26 In the proprietary model, software can also be shared within a firm but the firm's organizational boundaries plus code secrecy towards the outside limit the efficiency of this approach. Within a company hierarchy, a "culture of reuse and incremental improvement" (Cohen, et al. 2001, 4) is hard to adopt to an extent similar to open source.

4.2 Individuals and incentives

Although the body of FLOSS literature treats the question about motivation as a centerpiece, Rossi points out that an integrated, coherent explanation of the different types of incentives is still missing (Rossi 2004). I try to contribute by providing an extensive and structured overview of motivational factors drawn from work done over the last years (Weber 2004; Rossi 2004; Rossi and Bonaccorsi 2005; Earnshaw 2004).

I apply the standard distinction between intrinsic and extrinsic types of motivation (Ryan and Deci 2000). Extrinsic motivation refers to doing something because it leads to a separable outcome. This anticipated outcome may realize instantly or over time and can be positive (e.g., money) or negative (e.g., pressure). Table 4.1 gives an overview from the literature of the universe of extrinsic motivations of FLOSS developers.

Instant Benefits	*Delayed Benefits*
Low sharing costs v. high returns (Kollock 1999; Ghosh 1998; Lerner and Tirole 2002; Bonaccorsi and Rossi 2003)	Peer recognition/reputation (Dalle and David 2005; Lerner and Tirole 2001; Hars and Ou 2002)
Learning (Ye and Kishida 2003; von Krogh, et al. 2003; Lakhani and Wolf 2005)	Future career benefits, self-marketing (Lerner and Tirole 2004; Hars and Ou 2002)
"Scratch an itch"[27] (Lerner and Tirole 2004) Helping yourself and others (Weber 2004) User innovation (von Hippel 2005)	Reciprocity, "give and take" of code (Raymond 2001)
Monetary rewards (Hertel, et al. 2003; Feller and Fitzgerald 2002; Lerner and Tirole 2002)	"The joint enemy" (Weber 2004)

Table 4.1: Extrinsic motivation – Developing FLOSS as a means to gain other benefits (various sources)

In contrast, intrinsic motivation refers to doing something because it is inherently interesting or enjoyable. Inherent satisfactions can be fun or the challenge presented by an activity (Deci and Ryan 1985). Following Lindenberg, I divide intrinsic motivation into enjoyment- or obligation-based factors (Lindenberg 2001). Table 4.2 gives an overview from the literature of the universe of FLOSS developers' intrinsic motivations.

There is a common view that no single motivation stands out, "pro-social arguments" and "more narrow conceptions of individual benefits" are mixed (CED 2006). I like to point out the specific role of non-monetary incentives by which many of the developers seem to be driven.

[27] Attributed to Raymond 2001.

Enjoyment-based factors	*Obligation-based factors*
Hedonism, "fun" (Torvalds and Diamond 2002; Hars and Ou 2002; Lakhani and Wolf 2005)	Sense of community/identity[28] (Hars and Ou 2002; Weber 2004)
Art & Beauty, self-expression (Weber 2004)	Observance of norms (Hertel, et al. 2003; Zeitlyn 2003)
Ego-boosting, taking the challenge (Weber 2004)	Political mission (Stallman 1984; Raymond 2001; Hertel, et al. 2003)
Altruism, helping others (Hars and Ou 2002; Bitzer, et al. 2004; Zeitlyn 2003)	

Table 4.2: Intrinsic motivation – Developing FLOSS as an end in itself (various sources)

Empirical studies on the motivation (Ghosh, et al. 2002, Hars and Ou 2002, Hertel, et al. 2003, and Lakhani and Wolf 2005) paint a heterogeneous picture as well. No group of motivations stands out. While Hars and Ou stress extrinsic factors, Lakhani and Wolf find that enjoyment-based intrinsic factors important.

The studies, however, provide information on the composition of the community and the income mechanisms. Lakhani and Wolf state that less than 50% of the FLOSS participants were programmers; and only 1/3 of the respondents were employed to develop FLOSS. Moreover, Ghosh et al. mention that developing FLOSS "still resembles a hobby rather than salaried work. Besides (software) engineers and programmers, students play also a significant role in the community, but project performance and leadership is primarily a matter of professionals". They clustered the respondents in four categories, yielding the following distribution: 1/3 "Believers" (freedom/openness), 1/4 "Fun-Seekers" (intellectual stimulation), and 1/5 each in "Professionals" (job related) and "Skill-enhancers" (learning) (Ghosh, et al. 2002).

Note that only a minority does actually contribute source code (Rossi 2004). Many more provide general support to the project, writing documentation, helping users, etc.

To sum up, the software characteristics elaborated initially and the particular motivational setup of the FLOSS developers are the grounds, on which – as some suggest – a new model of innovation and production is about to emerge, one hardly possible in the pre-digital era. The final section presents the latest thinking on that proposal.

[28] Some of the principles of the FLOSS community, according to Weber, are: information should be "free", mistrust authority/no central control, judge people on what they create, create art&beauty on a computer, change human life for the better using computers (Weber 2004).

4.3 Community and innovation

Several efforts have been made to come up with a theoretical model of explaining how digital technology, motivational setup (and legal environment, see 5.1) sustain the open source innovation model.[29]

FLOSS innovation resembles the academic way of sharing and building upon the results of others rather than a market to sell goods (Lerner and Tirole 2004). Scholars have described this new model from various perspectives and in various terms.

Von Hippel and von Krogh talk about "private-collective models of innovation" (von Hippel and von Krogh 2003). They argue that programmers contribute freely to the provision of a public good because they garner private benefits from doing so. Seeing a direct private benefit influences the cost/benefit analysis of a single developer positively, whereas having private benefits and public interest in alignment sustains the system.

Not focused on FLOSS solely, von Hippel argues that users innovate more quickly and effectively than manufacturers if they are – legally, technically, and economically – able to, because they possess tacit knowledge of what their needs are. In addition, they do not have to make compromises for a whole market as the producer has to (von Hippel 2005; Chesbrough 2003).

Benkler identifies here a third model of production emerging besides capitalism and communism in what he terms "commons-based peer production" (Benkler 2002, 8).[30] He argues that technical change (moving to a digital area allowing identical copies at zero marginal costs) as well as economic changes (moving from an "industrial" to a "networked" information economy) are altering the way information is produced and exchanged. The networked environment enables a production mode, which is "radically decentralized, collaborative, and nonproprietary; based on sharing resources and outputs among widely distributed, loosely connected individuals who cooperate with each other without relying on either market signals or managerial commands." (Benkler 2006, 60).

To sum up, the open source model presents a system of producing and distributing software based on assumptions widely different to those of the established proprietary model. This raises the question whether traditional legal instruments, which aim to foster innovation, like patents, continue to do so under such a regime.

---

29 Although Shapiro and Varian have highlighted many characteristics of digital goods already back in 1999, all their remarks focused on the established, proprietary model (Shapiro and Varian 1999).

30 What a commons is and the reasons and consequences of FLOSS constituting a commons will be discussed in 5.1.

5 Patents and software innovation

5.1 Software as "intellectual property"

> *The dialectic of intellectual property rights is driven by the interaction of three conceptions; a pragmatic or economic point of view, a view that focuses on the property rights of creators, and a view that focuses on the uncircumscribed nature of ideas and the inherently communal nature of the creative process. The first point of view is the typical ideology of legislators, the second that of authors and publishers, and the third that of 'users'."*
>
> *Mitchell 2005*

In this first section, I describe how software is treated legally. After starting with a short history of software protection and a brief sketch of today's international "Intellectual Property (IP)" system regarding software, I draw a second line of distinction[31] between proprietary and free software based on the copyright particularities of the latter. Patents will be touched upon here but discussed extensively in chapter 6.

In the beginning, all software was open source and freely shared because it was not considered of value on its own. Users were programmers and software was simply considered an "instruction manual" to use mainframe computers. Software was shared and built upon and developers were paid for programming work. In the 1960s, the industry was still whole, not separated into hard- and software. Customers bought complete packages consisting of hardware, software, and support. Pressured by the U.S. Department of Justice for antitrust reasons, IBM decided in 1969 to separate the soft- and hardware business. This decision established the notion of software as a product on its own, known as "proprietary software", and permitted a software market to emerge in the 1970s (Grassmuck 2004).[32]

It is interesting to see how arbitrary the choice of legal protection for software appears when one looks at its history. It indicates that software is a difficult-to-seize subject for law. As a subject matter it stands out as one of the few that was considered for three different types of "IP" protection: sui generis[33], copyright, and patent law.

The first protection mechanism considered internationally was a copyright-inspired sui generis right with a shorter protection period. The World Intellectual Property Organization, a United Nations agency established in 1976, presented a treaty proposal in 1983. Hilty and Geiger assume that it was not adopted because just three years earlier the U.S. had voted in favor of the copyright mechanism for software protection. Lobbied by the U.S., other countries started accommodating their copyright

31 The first distinction is the innovation model as described in 4.

32 Microsoft was created in 1975, Apple in 1976. IBM entered the market for personal computers in 1981.

33 Lat. "*of its own kind*". A property right specifically designed for a certain subject matter. E.g. databases have an own protection mechanism in the EU.

regulation for software. The Community Directive[34] at European level (1991) and the WTO TRIPS[35] Agreement (1993) at international level solidified that move. Programs are since treated as literary works (Hilty and Geiger 2005, 619).[36]

Copyright protection is granted for the expression or form of an original literary or artistic work. Unlike patents, the underlying idea or concept is not protected and still accessible to others. This approach allows for independent development of software (expressed in different source code) for the same purpose (i.e., implementing the same algorithm). Yet, the suitability of copyright protection for software continues to be contested as well (e.g., Gordon 1998).

***

Triggered by the introduction of copyright protection, a second conception of software ownership, termed "free software", emerged in the '80s (Grassmuck 2004). A political movement formed that aimed at preserving the free sharing of software known from the early beginnings of the industry (Stallman 1984).

According to their definition, software is called *free* if it gives users the freedom to run, examine, modify, and redistribute programs without constraints. The technical access to the source code ("open source") is a precondition for these freedoms.[37]

To legally enshrine these freedoms, *copyright* is applied in an unorthodox way. A particular copyright license guarantees all these freedoms and requires that any further distribution of so-licensed software does not restrict these freedoms.[38] This last aspect is called *copyleft*: software that is built on top of free software and distributed needs to adhere to the same license conditions; the license conditions propagate.

As code ownership is communal and governed by a self-propagating license regime, the space of free software can be considered a *commons*. A commons is the space of intellectual material that is not subject to individual control but is instead controlled by a community. This is different to private property and different to the public domain, where material goes that is no longer protected by property rights (Boyle 2003).

---

34 Article 1 of the Council Directive on the legal protection of computer programs (91/250/EEC) states: "Member States shall protect computer programs[,] by copyright". (http://eur-lex.europa.eu, 08/21/2006).

35 Article 10 of the Agreement on Trade Related Aspects of Intellectual Property Rights (TRIPS) states: "Computer programs, whether in source or object code, shall be protected as literary works". (www.wto.org, 08/21/2006).

36 Article 4 of the WIPO Copyright Treaty states: "Computer programs are protected as literary works within the meaning of Article 2 of the Berne Convention." (www.wipo.int/treaties/en/ip/wct, 08/21/2006).

37 This definition is maintained by the Free Software Foundation (FSF 2006). The "open source" definition by the Open Source Institute (OSI 2006) is close in content albeit focuses more on practical aspects.

38 The "GNU General Public License (GPL)" (www.fsf.org/licenses/licenses.html, 01/23/2005) is used by the majority of free software projects. It is currently under revision and release of "GPL version 3" is scheduled for 2007.

Hess and Ostrom analyze information as a common-pool resource by considering five bundles of rights (Hess and Ostrom 2003, 124):

1. Access: the right to enter a defined physical resource system and enjoy nonsubstractive benefits;
2. Extraction: the right to obtain resource units or products of a resource system;
3. Management: the right to regulate internal use patterns and transform the resource by making improvements;
4. Exclusion: the right to determine who will have access rights and withdrawal rights, and how those rights may be transferred.
5. Alienation: the right to sell or lease management and exclusion rights.

The four freedoms put a program into a *software commons* and copyleft protects[39] it: Everyone is allowed to receive and use free software for any purpose (1/access). Copying is allowed for everyone without restrictions (2/extraction). Modification and redistribution are explicitly allowed (3/management). These rights are given to everyone through the copyright license, whose transfer is automatic by using free software – exclusion (4) is prevented. Finally, the most widely used license prohibits to make free code proprietary, which would be a precondition for being able to sell it.[40]

It is traditionally argued that property rights are created to prevent *free-riding* by internalizing external effects. Free-riding would reduce the incentive to innovate and the innovation rate would be suboptimal. But free software explicitly allows and encourages free-riding by re-establishing software's non-excludability characteristic (chapter 3). Consequently, the innovation model of open source can fully exploit the positive external effect that software is non-rival (chapter 4).

Technical progress is mainly driven by competition. Taking the above argument further, we can say that "IP" does not directly foster innovation but allows immaterial goods to become marketable and thus enables a market to play, in which competition can lead to innovation. At the same time, though, "IP" rights on technology can shift competition into imbalance or even monopoly, which gives rise to antitrust regulation (Heinemann 2002, 21-23). I will come back to the tension between free software as a "software commons" and patents as exclusive "IP" rights in chapter 6.

---

39 O'Mahony presents a variety of legal and other measures taken by free software projects to protect their work (O'Mahony 2003).

40 The "GPL" and "Lesser GPL" licenses prohibit and "BSD-type" licenses allow for appropriation. Jaeger and Metzger provide a thorough legal comparison of these and other licenses (Jaeger and Metzger 2002).

To conclude, free software can be considered as a freely provided complex public good (Bessen 2005) that is privately produced (Weber 2004) and self-protecting (O'Mahony 2003). Together with its innovation model, it is "an experiment in social organization around a distinctive notion of property rights." (Weber 2004, 227).

Before I discuss the relationship between patents and free software, I give a brief summary of what we know about patents in relation to proprietary software.

5.2 Advantages and disadvantages of patents

> *"According to some, granting patents for computer-implemented inventions stimulates innovation because the financial and material investment that is needed to develop sophisticated and specialised software is protected. - Others, however, believe that such patents stifle competition and act as a brake on innovation."*
> *(cii.european-patent-office.org)*

This section summarizes what we know about patenting of proprietary software, which has been the focus of most research so far. I start with a brief overview of the current policy situation in the U.S. and the EU. Then I define key terms and summarize the theoretical arguments. Finally, I discuss the small body of empirical literature.

USA. Driven by case law and changing practice of the U.S. Patent and Trade Office (USPTO) in the '80s, software and business methods have become patentable subject matters under U.S. jurisdiction. As a result, the number of related U.S. patents has increased significantly (Bessen and Hunt 2004). Jaffe identified three policy developments explaining the reasons for this change (Jaffe 2000, 532pp.): a) The creation of the Court of Appeal for the Federal Circuit (CAFC) was intended as a procedural reform that is "widely believed to have had a profound impact on the substantive outcomes of patent litigation", namely that the number of patents upheld in courts increased by over 50%; b) The circle of potential patent-holders grew by giving patenting privileges to federally funded institutions like universities and state laboratories; c) The scope of patentability was implicitly extended to software as courts upheld patents in a number of key decisions.[41]

EU. Similar to the USPTO, the European Patent Office (EPO) has gradually shifted its practice without backing from legislation. Hilty and Geiger note that despite the explicit exclusion of software in the European Patent Convention (Art. 52), the EPO has granted such patents "via the use of sophisticated legal constructions that are very hard, even for an intellectual property law expert, to comprehend." (Hilty and Geiger 2005, 620). They criticize that the legal uncertainty created was used by the European Commission to push for adjustment of legislation in the EU members states. With that

41 Often cited cases are *Diamond vs. Diehr*, upheld by the Supreme Court (1981) and *State Street Bank vs. Signature Financial Group*, upheld by the CAFC (1998).

aim, the European Commission issued a proposal for a *directive for the patentability of computer-implemented inventions* in 2002. That happened after a consultation phase, in which a lot of criticism was received about the lack of economic evidence justifying such a move.[42] After a much criticized process, the European Parliament rejected by a large majority the entire proposal in a second reading in June 2005.[43] Since then "the directive can be considered as dead and buried" (Hilty and Geiger 2005, 622).

Definitions. Lack of concise definitions and use of different legal terminology is a major hurdle when one tries to conceptualize and build an argument. I attempt to show that the two key terms in the legal debate – "software innovation" and "software patent" – are much less tangible than expected.

The three terms *idea, invention, and innovation* have specific legal meanings. An invention differs from an idea in that it must meet the "3-step-test" for patents. It has to be new, non-obvious to a lay-person, and susceptible to industrial application (see chapter 2). Software from our point of view fails the first two criteria: a) software's cumulative nature comes from the combination of a myriad of small ideas, whereas any single element is just too simple to qualify as an invention in legal terms; b) obviousness is relative: no matter how large and complex a big software system is, it is composed of reasonably-sized chunks completely understandable to skilled developers. As stated earlier (page 9), I follow Wheeler's definition that "software innovation" happens on the developer's side and constitutes a change in a programming paradigm rather than in mere source code.

There is no commonly accepted definition of what a *software patent* is. Not only the semantics of the term "software patent" are disputed but even whether the term itself is adequate. Allison and Lemley 2000 define them as "inventions solely embodied in software" (p10). That is no longer adequate, because legal practice in the U.S. has blurred this definition by creating a "doctrine of the magic words" (Cohen and Lemley 2001, 9): Patent specifications have been drafted to conceal the fact that they refer to software. Therefore I follow Bessen and Hunt's approach to define software patents as involving "a logic algorithm for processing data that is implemented via stored instructions; that is, the logic is not hard-wired" (Bessen and Hunt 2004, 8).[44]

---

42 For a discussion of the economic studies, see Hilty and Geiger 2005, pp.630.

43 The "software patent news" of the Foundation for a Free Information Infrastructure (FFII) provide a chronology of the process (http://press.ffii.org/Software_patent_news, 08/23/2006).

44 The directive proposal used the term "computer-implemented invention", which aims at looking at hard- and software separately. Still, the combination as a whole would be covered by a patent.

Arguments. Blind, et al. provide a synopsis of conceptual advantages and disadvantages of "intellectual property rights" for software based on an extensive literature review (Blind, et al. 2003, 11-34).[45] The list was cut down to the aspects relevant for patents. I annotate most arguments by a short comment and/or further supportive statements by others. I start with the potential advantages of software patents.

- *Disclosure fosters incremental and sequential improvement that increases diversity and interoperability*. This is true as long as the disclosed information is sufficient to allow replication and further improvement. As patent letters describe an invention in abstract terms – comparable to the specification language for software – they do not reveal *how* software works but merely describe *what* it does. In that regard, they do not provide helpful information for further development work.
- *Patents prolong the very short innovation cycles*. That is true from a company's perspective, I would not consider it an advantage for the economy. The main aspect is the granting process and duration of the patent itself.[46]
- *Patents direct resources to areas otherwise neglected*. Additional financial rewards may lead to searches for a broad application of a patent (Levine and Saunders 2004, 7). Patents may encourage developers to search for new applications of their software and hence to new innovations.
- *SME's and startups have a means to protect knowledge (their most valuable asset) as well as easier access to capital markets*. Patents can help (especially startups and SME's) to find funding by providing incentives to commercialize an invention. This lowers the hurdle for initial investments (Levine and Saunders 2004, 7).
- *Patents increase market transparency and decrease transaction costs*. Cross-licensing between companies can lead to transfer of knowledge otherwise impossible. Thus, patents may reduce transaction costs and uncertainty.
- *Patents improve knowledge flow through disclosure.* A patent requires public disclosure of the invention to support the invention's diffusion and avoid inefficient parallel development. Additionally, others are encouraged to

45 Please refer to chapter 2 for the definition of a patent.

46 Another delaying factor is backlog. In a press release of November 2005, the USPTO mentions a backlog of more than half a million patent applications. EPO patent examiners were on strike in May 2006 because of management plans to increase their "efficiency" in the light of an increasing backlog. See www.uspto.gov/web/offices/com/speeches/05-51.htm and www.heise.de/english/newsticker/news/73165 (05/26/2006).

engage in improving or substituting the invention. As "[s]oftware patents may prevent wasteful investments", more innovative activity can result (Levine and Saunders 2004, 7).

Next I summarize the arguments about the potential disadvantages of patents.

- *Patents constrict incremental and sequential development and thus reduce diversity and interoperability (especially open source), block new developments, and slow down innovation speed (Cohen and Lemley 2001).* Innovation can be slowed down or halted by a fundamental trade-off between generations of inventors (Scotchmer 2004): Given to the first inventor, a patent blocks the next. Given to the second, it does not support the first, who might not do the first invention. Innovation can be further slowed down by the administrative process of getting a patent. Further, patent duration is much longer as the software development cycle. Last, patent logic is based on the individual creator, not the collective, incremental innovation. "The conventional wisdom portrays manufacturers, and heroic inventors, as the source of innovations that are passively consumed by their customers" (CED 2006, 34) and patent logic reflects this view. In the case of FLOSS, however this assumption is not fulfilled. Bessen and Maskin argue that sequentiality and complementarity can increase firms' future profits. Patent protection may reduce overall innovation and social welfare (Bessen and Maskin 2000).
- *Patents constrict imitation and thus reduce the probability for sequential innovations and positive network effects*. a) Scotchmer criticizes economic incentive theory for patents as being too simplistic for the modern economy, not taking into account the cumulative nature of innovation and related externalities between generations of inventors who build on each other (Scotchmer 1991). b) Mitchell argues that patents are "the strongest form of intellectual property protection" as they protect the ideas themselves. In the case of software, "patented inventions cannot legally be reverse engineered" (Mitchell 2005, 29), which is impeding imitation. Even an independently developed algorithm would infringe a patent. c) Incentive theory states that assigning property rights fosters software innovation by preventing free-riding (Levine and Saunders 2004, 7). But free-riding in software fosters incremental and sequential innovation and FLOSS directly. Digital copies can be made easily and cheaply (Quah 2002).

- *Patents lead to inefficient allocation of resources: increased costs through alternative designs and shift of R&D expenditures to legal expenditures.* Ideas can be implemented and tested in software much faster and at much lower cost than is the case for physical inventions. This allows for much more test runs. This advantage is taken away by patents. First, by disallowing pursuit of alternative routes based on the same (patented) idea. Second, by increasing the legal costs connected to get, defend or challenge patents.
- *Patents lead to monopolies if network effects are present*. The software industry's market is more prone to monopolistic structures than others, because software shows strong positive network externalities (Katz and Shapiro 1985). The size of the effect is determined by the degree of interoperability. The more people use interoperable software to exchange data, the more useful that software becomes for them – no matter how innovative it is.[47] Consequently, most proprietary software companies follow a non-interoperability strategy for two reasons: a) to increase the customers' costs for switching to a competitor, which can lead to "vendor lock-in"; b) to lock out competitors who cannot easily develop interoperable software interfaces. In such an environment, first-mover advantages can create monopolies faster and easier than in other industries and lead to an "industry structure that is socially and economically not optimal" (Tuomi 2005).[48] Patents reinforce that tendency by preventing the development of interoperable programs and thus keeping switching costs high. Additionally, the incentive to "invent around" patented code may diminish interoperability as well, because it might not be possible to develop interoperable and yet non-infringing code. Non-interoperable software is perceived by users as of limited utility even if it is more innovative (Palmer 1989, 302).
- *Patents create legal uncertainties*. The logic of patents – protect the idea not the tangible form – makes it harder to decide in a specific case whether an infringement claim is valid or not. As the scope of patents are fuzzy by nature, so are decisions whether a program infringes or not.
- *Patents raise legal costs relative to production costs*. Digital copying lets the marginal costs of software production converge towards zero.[49] As a

47 Programs tend to be more interoperable if they stick to commonly agreed, open standards; and less if they are built on secret, proprietary standards.

48 Microsoft's monopoly can partly be explained by this characteristic of software.

49 The first instance of a program is the most expensive to produce, but every copy reduces the average production costs significantly. The marginal costs of each additional copy converges towards zero.

consequence, the marginal costs of exclusion (patents or copy protection) can be higher than the marginal costs of provision (coding and making the software available online). Spending resources to exclude non-purchasers would become inefficient (Palmer 1989, 275).

- *Patents constrict structurally more dynamic, small companies*. Patents put larger companies in a better position, as they have more resources to run a patent portfolio strategy (Blind, et al. 2003, 24).
- *Patent thickets emerge when many patents overlap* (FTC 2003, 6). A large number of patents reduce the overall amount of freely usable code for future innovations by "fencing" out code pieces and making them unaccessible to inventors. Underuse of available software functionality and less innovation can result – the "tragedy of the anticommons" (Heller 1997). This is particularly true for software, as implementing an idea in code can be done in a variety of ways, which only depend on the expertise of the developer. As a patent covers all "expressions" of an idea, many different solution paths (algorithms) can be barred by a single software patent.[50] In the case of FLOSS, patents may thus lead to under-provision of this public resource (Hess and Ostrom 2003, 128).

Empirics. As for the empirical evidence, the picture for the software industry is not conclusive. In a survey of 50 small software companies, Mann found that software patents have considerable value for established firms. He also found that they are of decreasing value, the younger the firm is, and that startups have little benefit from patents (Mann 2004).

Bessen and Hunt investigated which U.S. companies filed software patents from 1976-1999 and how their R&D expenditures developed. They found that a) granting of software patents did not lead to a rise in R&D investments; b) U.S. software patents are not used to enhance innovation per se but rather for "strategic patenting", i.e., to build up trading portfolios; c) patenting is not done by the software industry but by the computer, electronics and instruments industries. The dominant use of software patents for strategic purposes may lead to less innovation (Bessen and Hunt 2004).

This small review indicates that in the proprietary software industry no clear conclusions about the effectiveness of patenting on innovation can be drawn. FLOSS seems to be an even more different subject matter than proprietary software when discussing innovation and patenting.

---

50 As opposed to copyright which protects an "expression" but not the underlying "idea". Copyright is widely accepted today as a means to prevent software from being copied (Rossi 2004, 26).

6 Free software and patents

"In any discussion of information (including digital software) it is useful to remember that information is a human artifact (...) it is a "flow resource" that must be passed from one individual to another to have any public value."
Hess and Ostrom 2003

In this section, I present a new line of argumentation by connecting the specific motivational setup of FLOSS developers with some of the arguments given before.

Incentive theory assumes that exclusive property rights are needed to encourage innovative activity. But financial reward is just one of many possible motivations of a FLOSS developer; innovation still takes place. Then, "[h]ow does this new process fit with the conception of the innovative process driven by intellectual property rights (...) that we have inherited from Arrow and Schumpeter?" (Lerner and Tirole 2001, 821).

Schumpeter argues that companies innovate because they are seeking rents. For example, by bringing out new products regularly, they are able to keep ahead of competition (Schumpeter 1934). He also explains that the motivations to invest in the production process are not the same as those inspiring individuals to invent. The patent system serves to secure returns of the investor not the inventor. In FLOSS, there is no investor -or- inventor and investor are the same person.

Looking at the diverse range of intrinsic (Table 4.2) and extrinsic (Table 4.1) motivations that let developers engage in FLOSS innovation shows the following: out of 15 types of motivations only one is directly related to monetary rewards. Yet, the patent logic is solely based on the assumption that monetary rewards function as incentives to motivate people to invent. Patents only have a monetary value.

Based on this observation I propose a new line of argumentation. If we assume that developers are rational actors who try to maximize their utility when engaging in FLOSS development, we need to understand what they consider as utility and what increases or decreases their utility. What are their benefits and costs?

The *individual incentive of a developer to innovate* comprises two dimensions: the incentive to code and the incentive to commercialize code.

Incentive to code. Our central argument is that innovation in the absence of financial rewards still does occur because the FLOSS developers' *motivations* are different from that of developers employed in traditional proprietary software production settings. These motivational factors are predominantly non-monetary and are thus not affected by incentives induced by patents. [51]

[51] However, patenting might still lead to innovation. Developers may, for example, "work around" a patented functionality to build a non-infringing substitute. Having a free alternative can be a motivation, too.

Incentive to commercialize. Monetary investment necessary to contribute to FLOSS innovation is quite low: Besides a modest hardware infrastructure no expenses arise. Software material to build upon is freely available through the copyleft mechanism. Usually, FLOSS developers have no need to raise funds to contribute to or start a FLOSS project. What they do invest is their brains and their time.

The motivations of the developers together with the low initial investments may render the commercial incentive induced by patents ineffective. FLOSS developers do not seek profit from selling software and do not need to cover initial investments. Additionally, the free availability of code lowers the entry barrier for developers to such an extent that reuse of (modified) code in a new domain might happen easily; innovation takes place.[52]

The second argument is the *economic efficiency of the FLOSS market*. Two aspects are considered: Efficient diffusion of innovation and legal costs.

Efficient diffusion of innovation is driven by the openness of the code, the copyleft mechanism and the cost structure of software. The openness of the source code defies as unjustified the patent's requirement to publish how an invention works. Disclosure is an inherent characteristic of the FLOSS model. The copyleft mechanism enables collaborative approaches to innovation that can give an efficiency gain hardly possible in a competitive market setting. Finally, in contrast to proprietary software, FLOSS fully exploits its zero marginal cost nature. The last two aspects lead Benkler to call FLOSS a model of "commons-based peer production" (Benkler 2002, 8) that he considers a third mode of producing information goods, which is neither market-based nor hierarchical.

These aspects allow for a highly efficient online production system that lowers the costs to code (Benkler 2006) and can therefore stimulate innovative activity. Patents would induce inefficiencies into this system.[53] In addition, I argue that interoperability is given a higher priority in the FLOSS model than in the proprietary model because commercial competition is of no relevance. This improves the market efficiency by reducing the switching costs for users while preserving the positive network effects (Schmidt and Schnitzer 2003).

---

52 Two phenomena are: a) developers often imitate existing proprietary software to produce a free alternative; b) they take the code base of a FLOSS program as the basis for starting a new derivative ("forking").

53 In a more radical argument, the recent strand of radicalist/socialist theories (e.g., Boyle 1997) criticize the notion of the "romantic author" as socially constructed and argue that "all creations are the product of communal forces to some extent" (Menell 2000, 162).

Legal costs can be considered as inefficiencies in the FLOSS production system, because they increase transaction costs and potentially reduce innovation as some developers may avoid software domains covered by patents. I argue that patents burden developers with additional costs without providing them with effective, additional incentives to innovate. Herein, I follow Benkler's argument that assigning intellectual property rights would put inefficiencies in the distributed FLOSS development process and slow down innovation (Benkler 2002b, 81).

I argue that these two characteristics of the FLOSS innovation model make the threat of code underuse ("code-fencing", "tragedy of the anticommons") unlikely. Taking the previous arguments on market efficiency and code fencing together, I argue that FLOSS mitigates both threats to innovation through its free-sharing mechanism and that the patent disclosure mechanism is a less efficient alternative.

Patents do aggravate the *interoperability* issue. An interoperable program is more likely to infringe – and a non-infringing program is more likely to not be interoperable (and thus not competitive). This applies to proprietary software as well, but the danger is less as the source code is secret and cannot as easily be checked for infringement as open source code.

Patents may not be needed for open source development for two more reasons. a) patents allow for the distribution and coordination of development efforts among different companies. In free software development, efforts are already decentralized and software innovation happens to a large extent cumulatively. Patents are not needed to perform this function (cf. chapter 2); b) Different developers see different prospects to build on when confronted with an existing program. Pursuit of all directions of interest is given through the sharing convention. Patents on such prospects can cause high (social) costs by preventing developers pursuing different lines of development.

In summary, patents do not seem to fulfill all their objectives in connection with FLOSS. They guarantee "neither appropriate incentives nor social efficiency" (Quah 2002). But, FLOSS projects can file for patents and/or can be affected by patent claims of other parties. Data showing that patents are actively sought for would indicate that patenting as an instrument is useful and used in FLOSS projects. This will have to be compared with effects of third-party patent claims on code of FLOSS projects.[54]

Only empirical analysis can answer this question.

54 There is a (low) probability that FLOSS projects raise patent claims against other FLOSS projects. I have not heard of such a case yet.

7 Outlook

The primary research question addressed in this paper is whether and how patenting policy – the availability of patents and their enforcement – do affect FLOSS innovation at the individual developer level. My motivation lies in the expectation that depending on the motivations of FLOSS developers, certain arguments in favor of or against software patents play out stronger or weaker or are even neutralized.

This key difference between the proprietary and FLOSS model – the motivational setup – may lead to different effects of software patents on innovation under these two regimes. If this is the case, then a "one size fits all" approach to patent policy regarding software is not a sensible approach. No matter how it is designed, such an approach would systematically put one of the models at an advantage and the other at an disadvantage.

Today, FLOSS is no longer a niche phenomenon but an important market trend, as I showed in the introductory chapters. In addition, the sharing of "intellectual property" is slowly entering mainstream business logic.[55] This also shows the paradox situation the corporate world faces: Acquisition of increasing numbers of patents goes along with greater levels of "IP" sharing.

***

Theoretical considerations alone do not lead to a clear indication of whether a strong patent policy for software would increase or decrease innovation in the FLOSS domain. As empirical studies about this question are lacking, I intend to undertake a quantitative study with the following aims.

First, the hypothesis – that strong software patent policy has a decreasing effect on FLOSS innovation – needs to be tested empirically. I plan to run a survey among developers, particularly those with leading roles in FLOSS projects, asking them about their experiences with patents.[56]

A second aim of the study is to further illuminate the FLOSS innovation process and propose a more concise definition of what denotes *innovation in the FLOSS context*. I hope this to be a further step towards understanding the innovation process better and providing some groundwork to develop measurement indicators for it (df. Crowston, et al 2004).

A third aim, therefore, is to better understand the relationship between software patent policy and the motivational setup of FLOSS developers. Only if we understand

55 The Economist gave three reasons why sharing can be more profitable than "keeping it to yourself": growing complexity, convergence, disaggregation into niche firms (The Economist, 22 October 2005.)

56 A timely anecdotal example can be found at technocrat.net/d/2006/6/30/5032 (08/24/2006).

in a more coherent way what FLOSS innovation is and why it happens in the first place, are we able to argue about the adequacy of different incentive mechanisms.

***

As an outlook, I offer a few thoughts and questions for future research that lie beyond the scope of this paper.

Software is *legally* hard to conceive as it sits between artistic works and physical inventions. New conceptual ideas beyond copyright and patents may be worth considering. Hilty and Geiger discuss the question about adequate legal instruments to protect software by suggesting a sui generis model (Hilty and Geiger 2005). Open questions are: Should functional utility or originality (or something else) be used as basis for protection? How can the software characteristics and motivations of FLOSS developers by taken into account?

From an *economic* point of view, the costs and benefits of patenting for "FLOSS-based companies" is not yet much researched. Ideas, like protective patent pools, are being established to counteract the legal risk.[57] The principle is a pool of patents, pledged to not be used against FLOSS projects. The system is comparable to the "cold war"; the fear of retaliation keeps war absent – but does not necessarily mean peace. Putting potentially harmful patents under quarantine – ironically by putting them out in the open – is a costly process. This reduces the potential legal costs of patents, but how do patent-holders profit here? What is the rationale for having such patents when, as Benkler suggests, a shift towards a "commons-based peer production" (Benkler 2006, 8) is underway in the digital world? Another area profiting from more research are alternative incentive systems that are more in line with the FLOSS and not harmful to the proprietary model (FTC 2003, 46).

[57] A few examples are www.openinventionnetwork.com, www.patentcommons.org, www.osriskmanagement.com, www.pubpat.org and www.sflc.org(05/18/2006).

Bibliography

Allison J, Lemley M (2000). *Who's Patenting What? An Empirical Exploration of Patent Prosecution*. Vanderbilt Law Review, (53) , p. 2099.

Amabile TM (1996). *Creativity in Context*. Boulder, CO: Westview Press.

Arora A, Ceccagnoli M, Cohen WM (2003). *R&D and the Patent Premium*. NBER, Working Paper No. 9431. www.nber.org/papers/w9431

Arrow KJ (1962). *Economic Welfare and the Allocation of Resources for Invention*. In: Nelson R (Ed.), The Rate and Direction of Inventive Activity. New York: Princeton University Press.

Arundel A (2001). *The relative effectiveness of patents and secrecy for appropriation*. Research Policy, (30) , pp. 611-624.

Benkler Y (2006). *The Wealth of Networks*. New Haven and London: Yale University Press.

Benkler Y (2002b). *Intellectual Property and the organization of information production*. International Review of Law and Economics, (22) , pp. 81-107.

Benkler Y (2002). *Coase's Penguin, or Linux and the Nature of the Firm*. Yale Law Journal, (112) 369, pp. 1-79.

Bessen J (2005). *Open Source Software: Free Provision of Complex Public Goods*. Research on Innovation, Working Paper. www.researchoninnovation.org/opensrc.pdf

Bessen J, Hunt R (2004a). *The Software Patent Experiment*. Business Review. Federal Reserve Bank of Philadelphia, (Q3) , pp. 22-32.

Bessen J, Hunt R (2004). *An Empirical Look at Software Patents*. Federal Reserve Bank of Philadelphia, Working Paper No. 03-17/R. http://opensource.mit.edu/papers/ bessen.pdf

Bessen J, Maskin E (2000). *Sequential Innovation, Patents, and Imitation*. Research on Innovation/MIT, Working Paper. http://www.researchoninnovation.org/patrev.pdf

Bitzer J, Schrettl W, Schröder PJH (2004). *Intrinsic Motivation in Open Source Software Development*. FB Wirtschaftswissenschaft
FU Berlin, Working Paper No. 2004/19. http://opensource.mit.edu/papers/bitzerschrettlschroder.pdf

Blind K, Edler J, Friedewald M (2005). *Software Patents: Economic Impacts and Policy Implications*. Northampton, MA: Edward Elgar.

Blind K, Edler J, Nack R et al. (2003). *Software-Patente: eine empirische Analyse aus ökonomischer und juristischer Perspektive*. Heidelberg: Physica.

Bonaccorsi A, Rossi C (2003). *Why open source can succeed*. Research Policy, (32) 7, p. 1243–1258.

Boyle J (2003). *The Opposite of Property? (Foreword)*. Law and Contemporary Problems, (66) 1/2, pp. 1-32.

Boyle J (1997). *A Politics of Intellectual Property: Environmentalism For the Net?*. Duke Law Journal, (47) , pp. 87-116.

Boyles J (1996). *Shamans, Software, and Spleens: Law and the Construction of the Information Society*. Cambridge, MA: Harvard University Press.

Chesbrough HW (2003). *Open innovation : the new imperative for creating and profiting from technology*. Boston, MA: Harvard Business School Press.

Cohen JE, Lemley MA (2001). *Patent Scope and Innovation in the Software Industry*. California Law Review, (89) 1, pp. 1-57.

Cohen JE, Lemley MA (2001). *Patent Scope and Innovation in the Software Industry*. California Law Review, (89) 1, pp. 1-57.

Cohen WM, Nelson RR, Walsh JP (2000). *Protecting Their Intellectual Assets: Appropriability Conditions and Why U.S. Manufacturing Firms Patent (or Not)*. Cambridge, MA: NBER Working Paper 7552.

Committee for Economic Development (2006). *Open Standards, Open Source, and Open Innovation: Harnessing the Benefits of Openness*. Washington: www.ced.org.

Crowston K, Annabi H, Howison J et al. (2004). *Towards A Portfolio of FLOSS Project Success Measures*. In: Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Enginnering (ICSE 2004), Edinburgh, Scotland.

Dalle J, David PA (2005). *Allocation of Software Development Resources in Open Source Production Mode*. In: Feller J (Ed.), Perspectives on free and open source software. : MIT Press.

Deci EL, Ryan RM (1985). *Intrinsic Motivation and Self-Determination in Human Behavior*. New York: Springer.

Drahos (1996). *A Philosophy of Intellectual Property*1996. : Darthmouth.

Dutton (1984). *The Patent System and Inventive Activity During the Industrial Revolution 1750-1852*. : Manchester University Press.

Earnshaw NC (2004). *The Samba Project: Transformation of Self through Open Source Software Development*. Retrieved 21.06.2006 from http://samba.org/samba/news/articles/earnshaw_thesis.pdf.

European Information Technology Observatory (2006). *EITO 2006 Report*. Frankfurt: EITO.

European Information Technology Observatory (2004). *EITO 2004 Report*. Frankfurt: EITO.

Federal Trade Commission (2003). *To Promote Innovation: The Proper Balance of Competition and Patent Law and Policy*. Retrieved 27.04.2005 from www.ftc.gov/opa/2003/10/cpreport.htm.

Feller J, Fitzgerald B (2002). *Understanding open source software development*. Boston, MA: Addison–Wesley.

Fisher M (2005). *Classical Economics and Philosophy of the Patent System*. Intellectual Property Quarterly, (1) , pp. 1-26.

Free Software Foundation (2006). *Free Software Definition*. Retrieved 22.04.06 from http://www.fsf.org/licensing/essays/free-sw.html.

Ghosh RA (1998). *Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet*. First Monday, (3) 3, p. www.firstmonday.org/issues/issue3_3/ghosh/.

Ghosh RA, (ed.) (2005). *CODE : Collaborative Ownership and the Digital Economy*. Cambridge, MA: MIT Press.

Ghosh RA, Glott R, Kreiger B et al. (2002). *The Free/Libre and F/OSS Software Developers Survey and Study—FLOSS Final Report*. Retrieved 22.06.2006 from ww.infonomics.nl/FLOSS/report.

Goldman R, Gabriel R (2005). *Innovation happens elsewhere : open source as business strategy.*. Amsterdam: Elsevier.

Gordon SE (1998). *The very Idea!: Why Copyright Law Is an Inappropriate Way to Protect Computer Programs*. EIPR, (20) 1, pp. 10-13.

Grassmuck V (2004). *Freie Software – Zwischen Privat- und Gemeineigentum*. 2nd ed. Bonn: Bundeszentrale für Politische Bildung.

Harabi N (1995). *Appropriability of technical innovations. An empirical analysis*. Research Policy, (24) , pp. 981-992.

Hars A, Ou S (2002). *Working for free? Motivations of Participating inOpen Source Projects*. International Journal of Electronic Commerce, (6) 3, pp. 25-39.

Heinemann A (2002). *Immaterialgüterschutz in der Wettbewerbsordnung : eine grundlagenorientierte Untersuchung zum Kartellrecht des geistigen Eigentums*. Tübingen: Mohr Siebeck.

Heller M (1997). *The Tragedy of the Anticommons: Property in the Transition from Marx to Markets*. Harvard Law Review, (111) , pp. 621-688.

Hertel G, Niedner S, Hermann S (2003). *Motivation of software developers in the Open Source projects: An Internet–based survey of contributors to the Linux kernel*. Research Policy, (32) 7, p. 1159–1177.

Hess C, Ostrom E (2003). *Ideas, Artifacts, and Facilities: Information as a Common-Pool Resource*. Law and Contemporary Problems, (66) 1/2, pp. 111-145.

Hilty RM, Geiger C (20005). *Patenting Software? A Judicial and Socio-Economic Analysis*. International Review of Intellectual Property and Competition Law, (36) 6, pp. 615-754.

Hoppen N (2005). *Software Innovations and Patents - A Simulation Approach*. Stuttgart: ibidem.

Jaeger T, Metzger A (2002). *Open Source Software : Rechtliche Rahmenbedingungen der Freien Software*. München: Verlag C.H. Beck.

Jaffe A (2000). *The U.S. patent system in transition: policy innovation and the innovation process*. Research Policy, (29) , pp. 531-557.

Kahin B (2001). *The Expansion of the Patent System: Politics and Political Economy*. First Monday, (6) 1, p. -.

Katz M, Shapiro C (1985). *Network Externalities, Competition, and Compatibility*. The American Economic Review, (75) 3, pp. 424-440.

Klang M (2004). *Free software and open source: The freedom debate and its consequences*. First Monday, (10) 3, p. -.

Klincewicz K (2005). *Innovativeness of open source software projects*. Tokyo Institute of Technology , Working Paper. http://opensource.mit.edu/papers/klincewicz.pdf

Koch S (2005). *Evolution of Open Source Software Systems – A Large-Scale Investigation*. In: Proceedings of the 1st International Conference on
Open Source Systems, Genova, Italy.

Kollock P (1999). *The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace*. In: Smith M, Kollock P (Eds.), Communities in Cyberspace. London: Routledge.

König H, Licht G (1995). *Patents, R&D and innovation. Evidence from the Mannheim Innovation Panel*. Ifo-Studien, (41) , pp. 521-545.

Krishnamurthy S (2005). *An Analysis of Open Source Business Models*. In: Feller J (Ed.), Perspectives on free and open source software. Cambridge, MA: MIT Press.

Lakhani KR, Wolf RC (2005). *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. In: Feller J (Ed.), Perspectives on free and open source software. Cambridge, MA: MIT Press.

Lehmann M (2002). *Software Evolution*. In: Marciniak J (Ed.), Encyclopedia of Software Engineering. 2nd ed. New York: John Wiley and Sons.

Lerner J, Tirole J (2004). *The Economics of Technology Sharing: Open Source and Beyond*. NBER, Working Paper No. 10956. www.nber.org/papers/w10956

Lerner J, Tirole J (2002). *Some simple economics of open source*. Journal of Industrial Economics, (50) 2, p. 197–234.

Lerner J, Tirole J (2001). *The open source movement: Key research questions*. European Economic Review, (45) , pp. 819-826.

Lessig L (2002). *The Future of Ideas – The Fate of the Commons in a connected World*. New York: Vintage.

Levin R (1987). *Appropriating the Returns from Industrial Research and Development*. Cowles Foundation Paper, (714) , pp. 783-820.

Levine L, Saunders K (2004). *Software Patents: Innovation or Litigation?*. In: IFIP 8.6 Working Conference: IT Innovation for Adaptiveness and Competitive Advantage, Leixlip, Ireland.

Lindenberg S (2001). *Intrinsic Motivatoin in a New Light*. Kyklos, (54) 2/3, pp. 317-342.

Locke J (1690). *Two Treatises of Government, Book II. Of Civil-Government*. Project Gutenberg: www.gutenberg.org/etext/7370.

Lutterbeck B, Gehring R, Horns AH (2000). *Sicherheit in der Informationstechnologie und Patentschutz für Softwareprodukte – Ein Widerspruch? (Kurzgutachten für Dt. Bundesministerium für Wirtschaft und Technologie)*. Berlin: Forschungsgruppe Internet Governance.

Machlup F (1958). *An Economic Review of the Patent System. Study no. 15*. Washington, DC: U.S. Senate Judiciary Committee Subcommittee on Patents, Trademarks, and Copyrights.

Machlup F, Penrose E (1950). *The Patent Controversy in the Nineteenth Century*. Journal of Economic History, (10) , pp. 10-26.

Mann R (2004). *The Myth of the Software Patent Thicket*. American Law & Economics Association, Working Paper44. http://law.bepress.com/alea/14th/art44/

Marinova D, Phillimore J (2003). *Models of Innovation*. In: Shavinina LV (Ed.), The International Handbook on Innovation. Oxford: Elsevier.

Mazzoleni R, Nelson RR (2004). *Economic Theories about the Benefits and Costs of Patents*. In: Maskus KE (Ed.), The WTO, intellectual property rights and the knowledge economy. Cheltenham: Edward Elgar.

Menell PS (2000). *Intellectual Property: General Theories*. In: Bouckaert B, De Geest G (Eds.), Encyclopedia of Law and Economics. Cheltenham: Edward Elgar. Online at encyclo.findlaw.com.

Merges R (1997). *Intellectual Property in the New Technological Age*. New York: Aspen Publishers, Inc..

Mitchell HC (2005). *The Intellectual Commons: Toward an Ecology of Intellectual Property*. Lanham: Lexington Books.

O'Mahony S (2005). *Nonprofit Foundations and Their Role in Community-Firm Software Collaboration*. In: Feller J (Ed.), Perspectives on free and open source software. Cambridge, MA: MIT Press.

O'Mahony S (2003). *Guarding the commons: how community managed software projects protect their work*. Research Policy, (32) 7, pp. 1179-1198.

Open Source Initiative (2006). *Open Source Definition*. Retrieved 22.04.06 from http://www.opensource.org/docs/definition.php.

Palmer T (1989). *Intellectual Property: A nonPosnerian Law and Economics Approach*. Hamline Law Review, (12) 2, pp. 261-304.

Quah DT (2002). *Digital Goods and the New Economy*. CEPR, Working Paper No. 3846. cep.lse.ac.uk/pubs/download/dp0563.pdf

Raymond ES (2001). *The cathedral & the bazaar: Musings on Linux and open source by an accidental revolutionary*. Sebastopol, CA: O'Reilly.

Rossi C, Bonaccorsi A (May 2005). *Intrinsic vs. extrinsic incentives in profit–oriented firms supplying Open Source products and services*. First Monday, (10) 5, p. firstmonday.org/issues/issue10_5/rossi/index.html.

Rossi M (2004). *Decoding the »Free/Open Source (F/OSS) Software Puzzle« A Survey of Theoretical and Empirical Contributions*. University of Siena, Working Paper No. 424. http://www.econ-pol.unisi.it/quaderni.html

Ryan RM, Deci EL (2000). *Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions*. Contemporary Educational Psychology, (25) , p. 54–67.

Sakakibara M, Branstetter L (2004). *Do stronger patents induce more innovation? Evidence from the 1988 Japanese patent law reform*. In: Maskus KE (Ed.), The WTO, intellectual property rights and the knowledge economy. Cheltenham: Edward Elgar.

Sattler H (2003). *Appropriability of product innovations: an empirical analysis for Germany*. International Journal of Technology Management, (26) 5/6, pp. 502-516.

Saviotti PP (1997). *Innovation systems and evolutionary theories*. In: Edquist C (Ed.), Systems of innovation: technologies, institutions and organizations. London: Pinter.

Scacchi W (2003). *Understanding Open Source Software Evolution: Applying, Breaking, and Rethinking the Laws of Software Evolution*. Retrieved 21.06.2006 from opensource.mit.edu/papers/scacchi3.pdf.

Schmidt K, Schnitzer M (2003). *Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market*. Harvard Journal of Law and Technology, (16) 2, pp. 473-505.

Schumpeter J (1934). *The theory of economic development*. Cambridge, MA: Harvard University Press.

Scotchmer S (2004). *Innovation and incentives*. Cambridge, MA: MIT Press.

Scotchmer S (1991). *Standing on the Shoulders of Giants: Cumulative Research and the Patent Law*. Journal of Economic Perspectives, (5) 1, pp. 29-41.

Shadlen KC, Schrank A & Kurtz MJ (2005). *The Political Economy of Intellectual Property Protection: The Case of Software*. International Studies Quarterly, (49) 1, pp. 45-71.

Shapiro C, Varian HR (1999). *Information rules : A strategic guide to the network economy*. Boston, MA: Harvard Business School Press.

Stallman RM (1984). *The GNU Manifesto*. Retrieved 22.06.206 from www.gnu.org/gnu/manifesto.html.

Torvalds L, Diamond D (2002). *Just for fun: The story of an accidental revolutionary*. New York: HarperBusiness.

Tuomi I (2005). *The Future of Open Source*. In: Wynants M, Corneli J (Eds.), How Open is the Future?. : VUB Brussels University Press.

UNCTAD (2003). *E-Commerce and Development Report 2003. UNCTAD/SDTE/ECB/2003/1*. Retrieved 10.05.06 from www.unis.unvienna.org/unis/pressrels/2003/tad1967.html.

van Dijk T (1994). *The Economic Theory of Patents: A Survey*. MERIT Research Memorandum, (2) 17, pp. 1-39.

Viscusi W & et al (2001). *Economics of Regulation and Antitrust*. Cambridge, MA: MIT Press.

von Hippel E (2005). *Open Source Software Projects as User Innovation Networks*. In: Feller J (Ed.), Perspectives on free and open source software. : MIT Press.

von Hippel E, von Krogh G (2003). *Special issue on open source software development*. Research Policy, (32) 7, pp. 1149-1157.

von Krogh G, Lakhani K, Späth S (2003). *Community, joining, and specialization in open source software innovation: A case study*. Research Policy, (32) 7, p. 1217–1241.

Weber S (2004). *The Success of Open Source*. Cambridge, MA: Harvard University Press.

Wheeler DA (2001). *The Most Important Software Innovations*. Retrieved 26.02.2006 from http://www.dwheeler.com/innovation/innovation.html.

Wiener N (1961). *Cybernetics, or control and communication in animal and machine*. 2nd Ed. Cambridge, MA: MIT Press.

Wyatt S, Bertin G, Pavitt K (1985). *Patents and multinational corporations: results from questionnaires*. World Patent Information, (7) 3, pp. 196-212.

Ye Y, Kishida K (2003). *Toward an Understanding of the Motivation of Open Source Software Developers*. In: Proceedings of the 2003 International Conference on Software Engineering ICSE2003, Portland.

Zeitlyn D (2003). *Gift economies in the development of open source software: Anthropological reflections*. Research Policy, (32) 7, p. 1287–1291.