CIS

uni | eth | zürich
Center for Comparative and International Studies (CIS)

Working Paper No 40, 2009

Published by the Center for Comparative and International Studies (ETH Zurich and University of Zurich)

Hot Debate About Chilling Effects: Do Software Patterns Hamper/Free Open Source Software Development?

Marcus M. Dapp
Thomas Bernauer

ETH Zurich

Universität Zürich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Hot Debate about Chilling Effects: Do Software Patents Hamper Free/Open Source Software Development?

Marcus M. Dapp and Thomas Bernauer, ETH Zurich

Version: 20. February 2009

Abstract

The innovation-model in the free/open source software (FOSS) domain differs fundamentally from the innovation-model in the proprietary software domain. Many FOSS advocates claim that opportunities for software patenting, which have recently been expanded in some countries and are used primarily in the proprietary software realm, have negative effects on FOSS. This paper reviews the available evidence and concludes that we know surprisingly little about the empirical relevance of this claim. It argues that, if the claim holds true, negative effects of software patenting should be observable at the level of individual FOSS developers. We outline an explanatory model and research strategy to shed light on this question. The model specifies potential effects of patents on extrinsic and intrinsic motivations of FOSS developers, assuming that such motivations are necessary conditions for participation in FOSS projects and ultimately also innovation. Empirical testing of this model will have to be based on surveys administered to random samples of FOSS developers from different jurisdictions (with variation in software patent availability) and different domains of FOSS activity (with variation in "patent exposure").

Keywords: free software, open source, FOSS, motivation, innovation, patent.

1. Introduction

"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)

Computer software has, over the past few decades, become an ubiquitous and indispensable resource in all except the most impoverished economies. Two contrary developments have, since the 1990s, turned software development into a delicate subject for innovation policy. The first development is the tendency to extend and deepen the legal protection of software and other digital artifacts through copyright (Lessig, 2002) and patent policies (Bessen and Meurer, 2008) at national and international levels.[1] The intention of such policies is that temporary monopolies generated by patents (and copyrights) will allow innovators to appropriate the (monetary) benefits of their respective innovation. Conversely, the assumption is that in the absence of these monopolies, innovations would create excessive positive externalities (benefits primarily to actors other than the innovator) that would discourage investment in innovative activities.

The second development is the emergence and impressive growth of Free/Open Source Software (FOSS).[2] Viewed as an innovation system, FOSS has distinct characteristics that differ fundamentally from the common understanding of how software is produced and distributed: it is characterized by open access to and shared ownership of software code instead of proprietary code that is locked away. It is further characterized by self-governance and heavily decentralized organization instead of corporate hierarchy. And it is driven by a community of (often volunteer) developers who maintain and expand the code base instead of employees directed and paid by a firm.

Critics have attacked recent efforts to enhance opportunities for software patenting from two angles. Some argue that commercial software innovation differs from innovation activities in non-digital areas, and that software patents are at best inefficient and potentially even protectionist and obstacles to innovation. Other critics have pointed to "collateral damage" in the sense that patents on proprietary software could hamper innovation in the FOSS area (Free Software Foundation, 2008). While research on the first type of criticism has produced some, albeit still contested, results there is very little research on the second type of criticism. In view of the fact that a rapidly increasing number of companies, governments, non-profit

1 On software patenting in the USA see Jaffe (2000). For the EU, see Haunss and Kohlmorgen (n.d.) and http://ec.europa.eu/internal_market/copyright/documents/documents_en.htm (accessed on 15.10.2008). Efforts to introduce stronger legal protection are also manifest in international treaties, such as the Internet treaties of the World Intellectual Property Organization and the Trade-related Aspects of Intellectual Property Rights annex to the WTO agreement.

2 Besides the acronym FOSS, we use the two primary terms interchangeably, although open source (Open Source Initiative, 2006) usually stresses more the innovation system aspect, while free software (Free Software Foundation, 2006) stresses more the property rights perspective. Klang (2004) provides a detailed comparison.

organizations, and other actors are using FOSS[3] – examples include the Linux system, the OpenOffice.org suite and the Firefox browser– the need to fill this research gap is pressing.

The existing literature offers some insights into the determinants of FOSS developers' motivations (Krishnamurthy, 2006) and also the potential effects of software patenting on FOSS (Bessen and Hunt, 2007; Blind et al., 2005; Hoppen et al. 2003). However, to examine the effects of patenting on innovation in the FOSS domain we need to connect hitherto separate parts of the literature, particularly those on patents and on motivations. We submit that the most useful approach is to construct an explanation that accounts for individual FOSS developers' motivation, and to assess the effect of patents in that framework. The basic hypothesis to be tested is that, controlling for other factors that may influence the motivation of FOSS developers to engage in software innovation, patents have a negative effect.

The paper is structured as follows. We start by discussing the issue of patenting in traditional industries and the proprietary software industry. The following part examines the principal characteristics of FOSS in comparison to proprietary software, with an emphasis on the role of copyrights and patents and the underlying innovation model. We then outline three potential approaches to studying the effects of software patents on FOSS and focus on the third approach in the remainder of the paper. This approach illuminates whether participation in FOSS projects is negatively affected by patents. Building on the existing literature we outline an explanatory model that accounts for individual FOSS developers' motivations and place software patent availability in that model. The paper ends with suggestions for how the empirical relevance of the model and its hypotheses could be tested.

2. Do Patents Promote Innovation?

The characteristics of innovation processes differ strongly between traditional industries that produce physical goods and the software industry, which produces digital goods. These differences have important implications for arguments on the role of patents in promoting innovation.

Non-digital industries

"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)

A patent is a set of exclusive monopoly rights granted by the state to an inventor for a limited period of time (Scotchmer, 2004, ch. 3). 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. To qualify as such it has to be new, non-obvious, and suitable for industrial application. Once a patent is granted, all implementations

[3] See also European Information Technology Observatory (2004:128), Wheeler (2007), UNCTAD (2003), and OSOR (2008). Academic interest in the topic has increased as well. The new International Journal of Open Source Software & Processes will be launched in 2009. In 2003, the journal Research Policy published a special issue on FOSS (von Hippel and von Krogh, 2003b).

require permission of the patent-holder. In return, the patent-holder has to disclose the invention to the public, so that skilled persons can replicate it.

Studying the patent controversy in the mid-19th century, Machlup and Penrose (1950) outlined a typology of justifications for patents that is still widely accepted today (see also Fisher, 2005; Mazzoleni and Nelson, 2004; van Dijk, 1994; Heinemann, 2002).[4] The first justification, 'reward theory', holds that an inventor deserves compensation and reward for his up-front investment and risk, proportional to the usefulness of the invmention to society. The latter relates to the extent to which the invention allows social/economic actors to perform tasks better or satisfy needs more effectively and/or at lower cost. The second justification, 'incentive theory', is "probably the most quoted argument in favour of patents" (Dutton, 1984:20). It claims that patents promote innovation because, by preventing imitation, they motivate the individual to invent and commercialize inventions. In other words, patents increase the profit of the inventor and discourage competitors from free-riding. Because useful inventions increase society's welfare[5] and patents are inexpensive incentive providers, patents should be used to stimulate innovation (Fisher, 2005:14; Mazzoleni and Nelson, 2004; Merges, 1997; Campbell-Kelly and Valduriez, 2005). The third justification, 'exchange theory', argues that patents offer a fair balance between the public's and the inventor's interests. They encourage innovation and make the invention publicly available by requiring disclosure and motivating the inventor to commercialize the invention.

All three justifications argue hat patents motivate economic actors to innovate in ways that are also useful to society. Critics have persistently claimed, however, that patents are unnecessary. For example, they have noted that other types of rewards for innovators exist, for instance awards by private or public institutions (Menell, 2000). They have argued that inventions can take place independently, making disclosure likely because keeping an invention a shared secret without knowing from whom is difficult. And they have also pointed out that exploiting an invention without disclosing it is hardly possible (van Dijk, 1994).

Empirical research on whether patents promote innovation has produced mixed results. Patents appear to be most effective in promoting innovation in the drugs, chemical, and biotech industry, but seem to have little or no effect on innovation in other industries (Sakakibara and Branstetter, 2004; Cohen et al., 2000; Arora et al., 2003), particularly when compared to other strategies and mechanisms designed to commercially exploit inventions (Sattler, 2003; Arundel, 2001; Harabi, 1995; Bessen and Hunt, 2004a; Mazzoleni and Nelson, 2004; Heinemann, 2002).

4 We skip natural rights theory, one of the justifications listed by Machlup and Penrose, which uses a moral argument based on Lockean labor theory (Locke, 1690, Sect. 27; Drahos, 1996:43), and focus on economic justifications that focus on free-riding and preventing others from exploiting an invention without compensation.

5 Menell (2000:134) offers an interesting empirical analysis of the social value of innovation in the 19th century.

Proprietary software industry

"Information is information, not matter or energy." Wiener (1961)

The fact that software is digital has legal and economic implications. Software developers write so called source code. This code can be read by humans, but not by computers. Source code is translated into object code via compiler software. The end user only needs the object code, which can be read by computers but not humans, whereas software developers who seek to improve or modify software require access to the source code. At the source code level, it is often possible to achieve a specific program functionality via different solutions. This possibility is important for the discussion of legal protection of software. All countries that have joined the Berne Convention protect source code through copyright law. This implies that the copyright holder of a given piece of source code can decide how the software can be used. Owners of proprietary software normally allow customers to use the software but reserve all other rights on the software ("all rights reserved"). Consequently, end users of proprietary software have access to the object code but not the source code. In some jurisdictions, notably the USA and Japan, proprietary software developers can – in addition to copyright protection – also obtain patent protection on the functionality of software as implemented in its object code. That is, software developers in these countries can protect the source code through copyright and, in addition, functionalities in the object code through patents.

The remainder of this section discusses the principal characteristics of the proprietary software industry and the underlying innovation model. Based on this discussion we examine the role of patents in the subsequent section.

Software has rather pronounced public good characteristics because it is to a large degree non-rival and excluding people from using it is costly (Hess and Ostrom, 2003:119). Non-rivalry exists to the extent one person's use of a program does not limit or reduce the utility of this program to another person. In most cases the other person can simply copy the program, and the costs of doing so are usually very small. People can only be excluded from using a particular software if physical access is barred. Because software is an experience good exclusion can be counterproductive from the software producer's viewpoint: programs that are very difficult to get access to and use are of litte value. Exclusion also cuts against network effects (see also below), which are very important for the commercial success of software. Yet, making software easily available allows for free-riding. The software industry has used legal and technical protection measures to mitigate this problem (Quah, 2002).

The fixed costs of developing software vary considerably (Shapiro and Varian, 1999), but compared to industries producing physical goods the entry barriers in the software business are very low (Federal Trade Commission, 2003:45). Moreover, the fact that software can be copied at close to zero cost creates strong scaling effects and increasing returns to scale for producers. Scaling effects are strong because a producer can, within a very short time-period, scale (up, or down) production (Hoppen, 2005). Strong scaling effects also imply that a producer can reach a large market share in a rather short time (ibid.). This makes first mover advantages important and helps explain the financial success of many software firms (Cohen and Lemley, 2001:4; Blind et al., 2003).[6]

Another important characteristic of software is interoperability. Programs are called interoperable if they are technically capable of processing the same data formats or understand the same protocols. The value of using a given program increases, the larger the pool of users is with whom one can exchange data. Conversely, switching to another, non-interoperable program imposes considerable cost on customers. Interoperability thus creates strong positive network externalities (Quah, 2002) and can reinforce first mover advantages.

The proprietary software industry uses interoperability strategically to defend markets and lock in customers. For example, firms promote internally developed “back-box” data formats and standards in their programs to create barriers for competitors and increase switching costs for customers. Doing so creates and/or maintains a steady income stream as long as customers stay with the product – customers in fact end up using programs because many other people use them, too. For example, OpenOffice.org, a FOSS product, is widely considered sufficiently similar in functionality to the office suite of Microsoft, but potential users hesitate to switch because the long built-up pools of documents and peer users are vast. Similarly, many software firms try to attract new customers with a low-price strategy in the beginning. After a critical mass of users is reached and kept through non-interoperability strategies, network effects set in. Non-interoperability combined with scaling effects makes the size of the install base a critical success factor. For example, Skype’s proprietary protocols for internet telephony prevent Skype users from communicating with non-Skype users and thus lock out software suppliers using different protocols.

Yet another characteristic of software is functional utility. Software is useful because it processes information faster and more precisely than the human brain. Its value is derived from that problem-solving capability. Customers’ willingness to pay for that capability determines the price, rather than the property value of the software per se. This mechanism gives rise to differential pricing strategies, where the same program has different prices, depending on who the customer is (Shapiro and Varian, 1999). Utility is dynamic and tends to degrade over time because users’ needs change. At the point when a new program appears, its value is highest. Unless the software is adapted and upgraded, its utility decays and converges on zero when a better version becomes available. Marketing uses functional utility because selling software is difficult if the customer cannot see and feel the improvements. The more visible the changes, the easier it is to market the program. However, there is a problem of diminishing returns. A steeper learning curve can challenge users and harm adoption of the new version. Increased complexity (“feature bloat”) through more and more functions can reduce overall utility. Nevertheless, software prices are primarily justified by new functions and competition usually focuses on functionality and comfort.

---

6 Low entry barriers help explain why the European software market mostly consists of small- and medium-sized enterprises. Strong first mover advantages help explain why a few large US software companies, above all Microsoft, dominate the standard commercial software market.

Patenting of Proprietary Software

As shown in the two preceding sections the characteristics of software and its underlying innovation and business model differ quite strongly from the characteristics of non-digital goods. Do these differences imply that standard policy instruments designed to promote innovation, such as patents, are inappropriate for software. As shown by a recent statement by the European Patent Office this question is highly controversial:

"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." (http://cii.european-patent-office.org, accessed on 15.04.2007).

Lack of concise definitions and varying use of legal terminology are major hurdles when one tries to make sense of the controversy over patenting of software. To start with, the 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 appropriate for industrial application. Critics claim that software fails the first and second criterion: first, software's cumulative nature stems from the combination of a myriad of small ideas, whereas any single element is too simple to qualify as an invention in legal terms; second, obviousness is relative: no matter how large and complex a big software system is, it is composed of smaller-sized pieces that are easy to understand for skilled developers. The digital nature of software also blurrs the traditional invention-innovation distinction. Fagerbert notes that "[i]nvention is the first occurrence of an idea for a new product or process, while innovation is the first attempt to carry it out into practice" (Fagerberg, 2005:4). All digital goods are created through programs – including programs themselves. Software can thus be seen as the universal means of digital production, the "quintessential digital good" (Quah, 2002:29). It has the triple role of being the blueprint, the producing machine, and the final product all in one. From that perspective, the moment of invention and the moment of "physical manifestation" of that invention are identical, and innovation is the "first instantiation of a digital good" (Quah, 2002:7). Invention and innovation are thus, from the viewpoints of critics of software patenting, virtually the same.

Moreover, software patent is a term with no commonly accepted legal definition. Bessen and Hunt define software patents as covering "a logic algorithm for processing data that is implemented via stored instructions; that is, the logic is not hard-wired" (Bessen and Hunt, 2007:8). Allison and Lemley define the term in the sense of "inventions solely embodied in software" (Allison and Lemley, 2000:10). Legal practice in the United States, where software patenting is particularly pronounced, has evaded clear-cut definitions and has resorted to a "doctrine of the magic words" (Cohen and Lemley, 2001:9). Existing software patent specifications have been drafted in ways that conceal their reference to software. For purposes of empirical research (see further below) we will define software patent availability as the presence or absence of the possibility to obtain patent protection for software in a given jurisdiction. Availability is bound to a jurisdiction: a patent is only valid in those jurisdictions in which it has been applied for and granted.

Software patent availability becomes observable in two ways: for a software developer or company applying for patent protection, and when a software developer or company is confronted with a patent claim by another party.

The existing literature offers a considerable array of arguments for positive and negative effects of software patenting. We concentrate on the negative claims because this paper focuses on whether there are in fact negative effects.[7] The following arguments have been advanced against software patents (see, e.g., Federal Trade Commission, 2003; Blind et al., 2005; Blind et al., 2003:11-34; Levine and Saunders, 2004:7; Scotchmer, 2004; Committee for Economic Development, 2006:34).

The argument by proponents of patenting that patenting requires disclosure, and that disclosure fosters incremental and sequential improvement that increases diversity and interoperabilitym rests on the questionable assumption that the disclosed information is sufficient to allow for replication and further improvement. Patent language usually describes software inventions in abstract terms – much like language used to specify the design of a software. That is, it describes what the software ought to do, but does not reveal how it does so.

Innovation may be slowed down or halted by a fundamental generational trade-off between inventors: given to the first inventor, a patent creates obstacles for the next; and given to the second, it reduces incentives for the first, who may then not produce the first invention.

If patents prolong short innovation cycles this may be an advantage for a single company, but a disadvantage for the economy as a whole. In addition, the patent granting process tends to be slower than the typical software life cycle, so that the expected prolongation effect may in fact not be achieved.[8]

Patents restrict imitation and thus reduce the potential for sequential innovations and positive network effects. Scotchmer (1991), for instance, criticizes the application of economic incentive theory to patents as being too simplistic for the modern economy; he argues that it does not take into account the cumulative nature of innovation and related externalities and cumulative effects between generations of inventors. Mitchell argues that patents are "the strongest form of intellectual property protection" because they protect ideas themselves. In the case of software, "patented inventions cannot legally be reverse-engineered." This impedes imitation (Mitchell, 2005:29). Even independently developed algorithms can violate a patent.

7 The following arguments have been advanced in favor of software patents (Federal Trade Commission, 2003, ch. 5; Blind et al., 2005; Blind et al., 2003:11-34; Levine and Saunders, 2004:7): a) Patenting requires disclosure, and disclosure fosters incremental and sequential improvement that increases diversity and interoperability. It may also help in avoiding inefficient, parallel development of the same software by someone else. b) Patents prolong short innovation cycles and thus increase innovation pressure on companies. c) Patents direct investment into areas that would otherwise be neglected. They also motivate firms to seek broad application of a patent. d) Small and medium enterprises and startups can, through patents, protect valuable knowledge and obtain easier access to credit for further innovation and commercialization. e) Patents increase market transparency and decrease transactions costs, particularly in cross-license deals between companies.

8 In a November 2005 press release, the US Patent and Trademark Office mentioned a backlog of more than half a million patent applications. In May 2006, patent examiners of the European Patent Office were on strike because of management plans to increase their "efficiency" in the light of an increasing backlog.

According to incentive theory, property rights are assigned to prevent undesired free-riding, but free-riding (i.e., copying) can be advantageous in software innovation (Levine and Saunders, 2004:7).

If network effects are present, patents can favor monopolies. The software industry's market is more prone to monopolistic structures than other markets because software exhibits strong positive network externalities (Katz and Shapiro, 1985). The size of the network effect is determined by the degree of interoperability: programs tend to interoperate more efficiently if they adhere to commonly agreed, open standards; and less so if they are built on secret, proprietary standards. Most proprietary software companies follow a non-interoperability strategy that increases the customer's cost of switching to a competitor (see above). This can lead to 'vendor lock-in' and, conversely, lock-out of competitors who cannot easily develop interoperable software interfaces. Under such conditions 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, 2005:450). Patents can reinforce this tendency because they keep switching costs high and hamper independent development of interoperable programs. The incentive to "invent around" patented software code may diminish interoperability as well because it may not be possible to develop interoperable, non-infringing software code. Non-interoperable software is perceived by users as of limited utility even if it is more innovative (Palmer, 1989:302).

Patents may create legal uncertainties. They protect an idea in all possible forms based on a list of 'claims', which makes their boundaries fuzzy and the validity of a specific infringement claim difficult to decide. To judge which claims are violated – where infringement happened and to what extent – is usually decided by specialized courts. Generally, infringement decisions cannot be more clear-cut than the patent boundaries they are based upon (Bessen and Meurer, 2008, ch. 9).

Patents may be inefficient solutions for the problem of protecting property rights. Whereas the marginal costs of copying/providing software converge on zero, the costs of exclusion through patents, in contrast, are rather high. In many cases, therefore, exclusion costs will thus exceed provision costs and spending resources on excluding non-purchasers would thus be an inefficient investment (Palmer, 1989).

Smaller companies face disadvantages. Applying for a patent is costly in terms of the application process, managing the patent portfolio, handling license agreements, and litigation in case of infringements. Larger companies are therefore in a better position to run a patent portfolio strategy because they have more resources (Blind et al., 2003:24).

Many patents, notably when overlapping, may create "patent thickets" (Federal Trade Commission, 2003:6). A large number of patents reduces the overall amount of knowledge usable for future innovations by "fencing" out pieces and making them inaccessible. Underprovision of knowledge can result in less innovation – the "tragedy of the anticommons" (Heller, 1997). This may be particularly important in the case of software, because implementing an idea in software can be done in a variety of ways that only depend on the expertise of the developer. Since a patent covers all ‘expressions’ of an idea, many different solution paths (algorithms) can be barred by a single software patent. In contrast, copyright, which is widely accepted “as a means to prevent software from being copied” (Rossi, 2004:26), protects a specific expression (source code) but not the underlying concept and functionality of a particular software.

While the list of potentially negative effects of software patenting that one encounters in the relevant literature is longer than the list of potentially positive effects (the latter are used to justify patenting) the empirical evidence is – surprisingly – very thin. In a survey of 50 small software companies, Mann found that software patents are of considerable value to established firms and that they are of decreasing value, the younger the firm is. Startups hardly benefit from patents (Mann, 2004). Bessen and Hunt (2007) identified algorithmically 130,650 software patents that were granted to US companies in 1976-1999. They found that differences in software patent propensity across different parts of the software industry are large. Propensity is not highest in the software publishing industry as one might have expected, but in the electronics and computer industry. They also found that “the very large increase in software patent propensity over time is not adequately explained by changes in R&D investments, employment of computer programmers, or productivity growth” (ibid.:1). In other words, existing studies do not tell us much about whether software patents have positive, negative, or no effects on innovation in the proprietary software industry.

3. Free/Open Source Software (FOSS)

"In any discussion of information (including digital software) it is useful to remember that information is a human artifact (...)
a 'flow resource' that must be passed from one individual to another to have any public value."
Hess and Ostrom(2003:131)

We now examine how free/open source software (FOSS) differs from proprietary software. We show that FOSS represents a process innovation system in software development that exploits basic software characteristics differently than the proprietary system.[9] This analysis leads to a discussion of potential effects of software patenting on FOSS.

Free/open source software (FOSS) differs in important ways from proprietary software. Proprietary software is protected through restrictive copyrights on source code (users are allowed to use the object code, but do not even have access to the source code) and in some cases also patents on functionalities expressed in the object code. In stark contrast, users and developers of FOSS obtain a much greater set of rights, in particular the right to copy, modify and pass on source code as well as object code. Patents on FOSS software are possible, depending on the jurisdiction, but are extremely rare because they are widely regarded as incompatible with the innovation paradigm of the FOSS community.

Sharing of source code is the principal social activity in the open source innovation system. The continued access to source code – a key prerequisite for sharing – is the backbone of this system. Benkler argues that the personal computer as a cheap, universal means of production and the internet as an ubiquitous and cheap means of many-to-many communication has removed the physical constraints on information production that required market-based strategies based on exclusive rights to undertake the high investments needed. Declining infrastructure costs allow non-market, non-proprietary production of information goods through "coordinate{d} effects of the uncoordinated actions of a wide and diverse range of individuals and organizations acting on a wide range of motivations" (Benkler, 2006:4-5).

Decentralized, large-scale collaborative software development is the primary activity of the open source system. Developers join virtual communities that gather around software projects hosted on openly accessible websites. The internet allows for fast communication and coordination and represents a low entry barrier for new contributors. Code is written, copied, and recombined with other code, while access to all code is legally guaranteed. Very large projects often establish nonprofit foundations that can assume a variety of protective roles (O'Mahony, 2005).

Giving source code away without charging royalties is the 'commercial' activity of the open source system. Programs could be sold, but this is usually not practiced among FOSS community members because further distribution for free would be allowed anyway. Much weaker market forces are at play, however,

9 Amabile (1996) defines innovation as the implementation of ideas through social, commercial or organizational activities. Showing distinct activities in all three dimensions, the FOSS innovation system itself can be considered a process innovation.

because traditional royalty-based producer-consumer-relationships do not exist. Consequently, a project's success can only partially be measured in commercial terms.[10] While open source projects compete on the technical merits of the software and the attraction of capable developers, companies engaged in open source development continue to compete in the typical services of the software business: maintenance, customization, support, and training. Several companies (e.g., IBM, HP, Sun, Red Hat or Canonical) have built business models in line with the open source development model. Producing software in a collaborative manner and giving it away for free is, therefore, not incompatible with a software business (see Goldman and Gabriel (2005) and Krishnamurthy (2005) for analyses of FOSS business models).

In managing four characteristics of software in a manner that differs from the proprietary software industry, the open source system uses a different innovation mechanism. (1) The positive network externalities and public goods character of software are viewed as an advantage rather than a problem because FOSS is deliberately made non-excludable. On the demand side, various factors encourage the wide distribution of programs: low costs, a permissive copyright regime, and the adaptability of the software to each user's own needs. On the supply side, many users make a project large and visible and help attract more developers, supporting further project growth. (2) Developing software primarily based on technical considerations is easier than under additional market pressures. That is, functional utility rather than marketing considerations are at the center of development efforts. (3) For the same reason, developers can pursue interoperability as a technical goal, instead of trying to lock in customers. Interoperability makes code sharing easier, helps programs to interact more effectively, and reduces switching costs. (4) FOSS life cycles tend to be even shorter than in the proprietary system, because development happens incrementally, with many more small changes and shorter release periods than in the proprietary system. With no time-to-market pressure and no market-driven deadlines to meet, code can be tested more thoroughly before release.

The core activities in FOSS development – sharing code, giving it away for free, and collaborating in "virtual" and heavily decentralized communities – lead to two characteristics that are unique to FOSS. These can be described using Saviotti's evolutionary innovation model that analyses innovations as mutations that generate variety (Saviotti, 1997; Marinova and Phillimore, 2003).

First, code variety in FOSS tends to be greater than in the proprietary software realm. This means that a greater diversity of solution paths to solve a specific problem is pursued than in the proprietary system. Complementary innovation, i.e. unrestricted numbers of independent parallel pursuits, may achieve an innovation goal with higher speed and probability – if solutions are shared and accessible to all (Bessen and Maskin, 2000). Greater variety in solutions occurs because the large numbers of users' needs and developers' interests lead to more heterogeneity (Bessen, 2005): different developers see different prospects to build on when confronted with a certain programming task. If no consent on the overall direction of a project can be

[10] Scotchmer (2004, ch. 2) offers an economic analysis of why public goods should be provided for free.

found, a split – a so called forking – usually occurs. Project forking adds to the variety of approaches to solve a given problem, but also splits and therefore weakens community resources.

Second, FOSS development is strongly cumulative and an example for reproduction and inheritance in the evolutionary model: various developers make many small changes and recombinations, thus building up the code base over many iterations, to expand its functionality and constantly adapt it to their own problems (Quah, 2002:29). The complementary and cumulative nature of innovation, particularly in high-tech industries, is widely recognized (Hoppen, 2005). The concepts of incremental (Scotchmer, 1991) and sequential (Bessen and Maskin, 2000) innovation are based on the same idea, through they stress different aspects. A software program may grow to a level of complexity where a single person cannot understand the whole anymore. Yet code modularization enables a skilled programmer to continue contributing. Consequently, even large groups of isolated individuals can effectively collaborate in large systems. In the proprietary system, software can also be shared within a firm, but the firm's organizational boundaries and code secrecy towards the outside limit the efficiency of this approach. Within a company hierarchy, a "culture of reuse and incremental improvement" is much harder to implement (Cohen et al., 2000:4).

Several studies offer theoretical models to explain how decentralized innovation, motivational setup, and legal regimes sustain the open source innovation system. They tend to view FOSS innovation as resembling the academic way of sharing and building upon the results of others rather than a market in which goods are sold (Lerner and Tirole, 2004). Von Hippel (2005) argues that users innovate more quickly and effectively than manufacturers if they are – legally, technically, and economically – enabled to, because they (tacitly) know best what their needs are. Unlike the producer, they do not have to make compromises for a diverse market (Chesbrough, 2003).Von Hippel and von Krogh (2003a) have developed a "private-collective model of innovation". They argue that programmers contribute freely to the provision of a public good because they garner private benefits from doing so. Direct private benefits improve the cost/benefit analysis of the single developer, and aligned private benefits and public interest sustain the system. Benkler notes that technical (moving to a digital environment) as well as economic changes (moving from an industrial to a networked information economy) have altered the way in which information is produced and exchanged. He calls this third model of information production – besides market-based capitalism and central-planning communism – "commons-based peer production" (Benkler, 2002:8). The network enables a production mode that 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).

In brief, the FOSS system deals with production, ownership, and distribution of software in ways that differ fundamentally from the proprietary software system. FOSS is a freely provided complex public good (Bessen, 2005) that is privately produced (Weber, 2004) and self-protecting (O'Mahony, 2003). It can in fact be regarded as "an experiment in social organization around a distinctive notion of property rights." (Weber, 2004:227).

4. Do Patents Affect FOSS?

The principal justification for software patents is that they encourage innovation in the proprietary realm of software development. Does this imply they are simply irrelevant to innovation in the FOSS realm because the FOSS innovation model operates in a very different mode? Or do patents generate, as critics argue, negative spill-over effects from the proprietary to the FOSS innovation system? If so, are such effects similar in nature to the ones critics have voiced with respect to the effects of patents in the proprietary software area? How can we study this claim empirically to establish whether the critics are right? We can think of at least three potential approaches to studying the effects of patents on FOSS empirically.

First, we could identify whether FOSS developers have in fact become targets of litigation over infringements on software patents. In principle, patents can affect the proprietary and the FOSS domain, but patent violations are probably easier to prove in the FOSS domain, where the source code can easily be inspected.[11] We do not know of any systematic studies of this first type. However, even if they existed they could not illuminate the "true" nature of the effect of patents. If, for example, we observed 100 cases of litigation we would still not know whether these are the tip of the proverbial iceberg – FOSS developers might be bullied by patent holders to an extent that formal litigation is unnecessary, or they might simply stay away from areas where patent density is high – or just a drop in the ocean of FOSS developer activity.

Second, we could examine whether, controlling for other determinants, FOSS innovation is slower or smaller in areas where patenting in the related proprietary software domain is "thicker". Such a study would capture the aggregate net effects of software patenting. It could be implemented in the form of a global comparison across specific types of software and/or over time. We do not know of any studies of this second type. One of the main difficulties with this approach is to define the dependent variable (innovation) in aggregate terms that would be empirically useful across different types of software as well as the proprietary and FOSS area. This takes us to the third approach where we argue that effects of patents should be observable when we study innovation from the perspective of the individual developer.

Third, we could examine whether participation in FOSS projects is negatively affected by patents. In other words, the hypothesis in need of testing is that, all other influences held constant, participation in FOSS projects is negatively affected by patents. To test this hypothesis, we need to start with a baseline model that accounts for developer participation in FOSS projects. We think that the third approach is the most feasible and useful one and hence focus on this approach in the remainder of the paper.

Motivations for participation in FOSS projects

Several studies have recently examined the driving forces of participation in FOSS projects. Rossi (2004) points out, however, that an integrated and coherent explanation of the different types of incentives is still missing. Krishnamurthy (2006) provides an overview of several empirical studies of this kind. He observes

11 See, for example, www.groklaw.net and www.chillingeffects.org about (as of 21.07.2008).

that "…both intrinsic and extrinsic motivational components are important and do exist…" and that "…the evidence is mixed on the relative value of intrinsic and extrinsic motivational components…" (ibid., 27). Ryan and Deci (2000) argue that intrinsic motivation is present when the respective actor behaves in a certain manner because such behavior is inherently interesting or enjoyable, for instance because of the fun or the challenge it involves. Extrinsic motivation is present when the respective actor behaves in a certain manner because such behavior leads to a separable outcome – instantly (reward) or with a time delay (incentive). Drawing on the framework of Rossi and Bonaccorsi (2005), we submit that research should consider the motivational factors summarized in Tables 1 and 2. Table 1 offers an overview of extrinsic motivational factors. It will be noted that only two of the eight factors are monetary ([M]) in nature.

Table 1: Extrinsic motivations

Rewards (instant)	**Incentives (delayed)**
Learning of new skills (Ye and Kishida, 2003; von Krogh et al., 2003; Lakhani and Wolf, 2005)	Expecting others to give back, reciprocity (Raymond, 2001)
Helping yourself by developing own solutions (Weber, 2004; Lerner and Tirole, 2004; Raymond, 2001; von Hippel, 2005)	Peer recognition, reputation (Dalle and David, 2005; Lerner and Tirole, 2001; Hars and Ou, 2002)
Low sharing costs compared to return of code shared by others (Kollock, 1999; Ghosh, 1998; Bonaccorsi and Rossi, 2003)	[M] Future career benefits through self-marketing (Lerner and Tirole, 2004; Hars and Ou, 2002)
[M] Direct monetary reward, income (Zeitlyn, 2003; Feller and Fitzgerald, 2002)	Fighting proprietary software, the 'joint enemy' (Weber, 2004)

Table 2 offers an overview of intrinsic motivational factors. Drawing on Lindenberg (2001) we can distinguish them further into enjoyment- and obligation-based factors.

Table 2: Intrinsic motivation – Developing FOSS as an end in itself

Enjoyment-based factors	**Obligation-based factors**
Fun, hedonism (Torvalds and Diamond, 2002; Hars and Ou, 2002; Lakhani and Wolf, 2005)	Identification and sense of community (Hars and Ou, 2002; Weber, 2004)
Self-expression, 'coding as art' (Weber, 2004)	Observance of community norms like, e.g., sharing (Zeitlyn, 2003)
Helping others, altruism (Hars and Ou, 2002; Bitzer et al., 2004; Zeitlyn, 2003)	Political mission, 'software must be free' (Stallman, 1984; Raymond, 2001)
Ego-boosting through solving difficult problems, challenge (Weber, 2004)	How one is viewed by significant others, e.g. family, friends (Hertel et al., 2003)

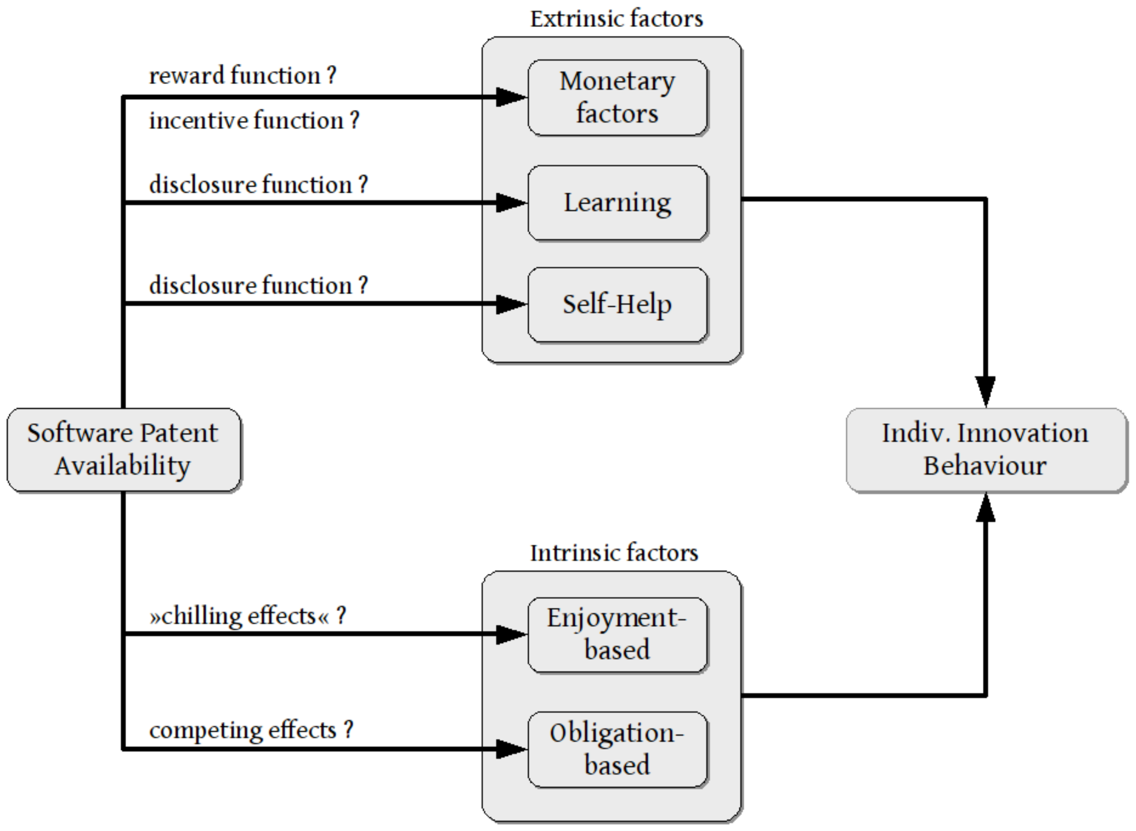
Empirical studies offer considerable support for the influence of the factors listed in Tables 1 and 2 (e.g., Ghosh et al., 2002, Hertel et al., 2003). However, the list is broad and diverse – empirically, no single motivation appears to stand out as particularly important. In addition, the relative importance of extrinsic vs. intrinsic factors remains unclear. Hars and Ou (2002) identify effects of extrinsic factors, Lakhani and Wolf (2005) find that intrinsic factors are important, whereas Roberts, et al. (2006) find no impact of intrinsic motivation[12]. By implication, the findings with respect to "social arguments" vis-à-vis "more narrow conceptions of individual benefits" are mixed (Committee for Economic Development, 2006:22).

Effects of patents

How could we bring the software patent issue into a model accounting for participation in FOSS development? There are two possibilities. First, we could simply view the explanation of participation in terms of an additive, linear process, in which software patent availability influences the propensity of individuals to participate in FOSS development alongside the intrinsic and extrinsic factors discussed above. Second, we could view the effects of software patents in terms of effects on motivations to participate in FOSS development. We believe that the second option is more appropriate. Figure 1 summarizes the main components of such a model. The remainder of this section will discuss the principal variables and causal effects in more detail.

Figure 1: Effects of software patent availability on participation in FOSS development

12 Roberts, et al. (2006) studied three main web server related projects of the Apache Software Foundation that are commercially very important. The largely commercial nature of this project may have influenced the findings.



The research approach we suggest concentrates on innovation from the developer rather than the user perspective. It does so for several reasons. Proprietary and FOSS systems differ not only with respect to the process of innovation (how software is developed), but also with respect to the product innovation dimension. The proprietary system usually applies a user-centric innovation perspective: "new" means new for the user. However, a user may perceive a new software feature as new even when the underlying source code is not new from a developer's perspective. Conversely, two software programs performing the same task may do so in different ways and, even if one implementation is more innovative than the other, a user may not recognize the difference. From this perspective, the argument by Klincewicz (2005) that most FOSS projects are 'me-too' clones of existing proprietary programs and are not innovative on their own appears questionable (Wheeler, 2001).[13]

Wheeler (2001) argues that most FOSS activity at the source-code level consists, much like in the proprietary software domain, of recombination or integration of existing components and is not innovative. He proposes a definition under which only new programming paradigms qualify as innovations (ibid.). This very demanding definition allows for only a few innovations per decade and is not very useful for our

13 Generally, software innovation stems from two equally important sources: competition of independent ideas, i.e., different, independent paths can all lead to competing programs with equivalent functionality; imitation, i.e. building on and extending previous work. The FOSS system utilizes imitation more extensively than the proprietary system (Bessen and Maskin, 2000).

purposes. A more practical approach is to relate individual code contributions to specific levels of innovativeness. At the low end of the innovation scale, we can place unaltered use of libraries and reuse of code fragments with varying degrees of adaption. At the high end, we can place new implementations of existing algorithms or newly devised algorithms.[14] Table 3 sketches a set of ordinal categories to that end.

Table 3: Levels of code contributions as a proxy for FOSS innovation

Innovation level	**Type of code contribution**
5	inventing new algorithms/methods before coding (»algorithm II«)
4	coding of known algorithms/methods from scratch (»algorithm I«)
3	recombining existing FOSS components with much adaption (»reuse II«)
2	integrating existing FOSS components with little adaption (»reuse I«)
1	linking to existing FOSS libraries (»library«)
X	reverse-engineering/imitating functionality from non-FOSS programs

Note: reverse engineering is a specific way of producing code and should be considered separately.

Innovation behavior is affected by two types of motivational factors (extrinsic and intrinsic), as discussed above. We can, as a starting point, simply assume that any type of motivation has a positive effect on the frequency and extent of innovation behavior. A full model would, of course, also have to include a set of control variables (e.g., skills, income, age, number of other developers in the project).

Patent availability can be measured both in subjective and objective terms. Subjective measures will have to rely on surveys of FOSS developers, for example asking them about the application domains (for example multimedia, security, office, etc.) in which patents are particularly prevalent and strong. More objective measures could ask developers whether they are based in a country with strong or weak opportunities – both in terms of the laws and patent office behavior – for patenting software. Most FOSS projects only exist in cyberspace and have no legal representation through a foundation, company or association within the boundaries of a particular country. Each contributor, however, is resident in a jurisdiction with particular rules and practices on software patenting. We can therefore examine whether patent effects differ depending on the jurisdiction out of which the developer operates.

Whether and to what extent software patent availability affects extrinsic and intrinsic motivations in positive or negative ways is empirically open, and no research on this issue exists. As depicted in Figure 1 we suggest three extrinsic and two intrinsic motivational factors and discuss in the remainder of this section how patent availability might affect these factors and hence also participation in FOSS projects. Table 4 summarizes arguments for the case when a FOSS developer faces a software patent claim from someone else.

[14] This point touches on the discussion raised by Klemens (2005) whether algorithms, which are essentially math, should be patentable subject matter at all.

Table 4: Potential effects of (facing) software patents on extrinsic and intrinsic motivations

Motivation Factor	Positive effects	Negative effects
Extrinsic (instant):		
Earn money/income (monetary)	none	(-) Legal defense costs reduce income
Learn skills (disclosure)	(+) May reveal useful knowledge.	(-) Knowledge revealed is insufficient
Help yourself (disclosure)	(+) May reveal useful knowledge.	(-) Knowledge revealed is insufficient (-) Legal risk to include code
Net gain of code	none	(-) Legal risk to include code
Extrinsic (delayed):		
Future career (monetary)	none	(-) May threaten project, in which developer engages to demonstrate his skills
Intrinsic (enjoyment-based):		
Joy	none	(-) Legal risk reduces fun
Altruism	none	(-) Legal risk increases costs of altruistic behavior
Artistic self-expression	none	(-) Limits self-expression in writing code
Solve difficult problems (ego-boosting)	none	(-) Limits options when writing code
Intrinsic (obligation-based):		
Observance of community norms	(+) threat strengthens community, "rally round the flag" effect	none
Identification with community	(+) threat strengthens community, "rally round the flag" effect	none
Software freedom	(+) Directed efforts to circumvent patents may lead to innovations	none

Note: other types of motivations listed in Tables 1 and 2 but not in Table 4 are less likely to be affected by patents. Hence we omit them here. Such motivations include, notably, (extrinsic) reciprocity, peer recognition, fighting proprietary software, (intrinsic) opinions of significant others about someone's FOSS engagement.

Software patents are likely to have negative effects on all extrinsic factors, though effects on learning and self-help are perhaps less clear because they depend on whether the knowledge embedded in a patent is accessible for replication. The lack of source code in patent letters is one of the problems in this regard. As discussed above, knowledge in the FOSS system is publicly available down to the source code level, whereas the typical (software) patent language is intentionally kept broad and abstract and source code is kept secret.

As to the intrinsic motivations, enjoyment-based factors may be reduced by the availability of software patents, whereas obligation-based factors may be strengthened when developers face patents. The negative effect on enjoyment-based factors is likely because patents reduce the freedom of action of FOSS developers. Legal risk and potential legal costs are also likely to reduce fun and altruistic behavior. Self-expression means that writing FOSS code is perceived as an art: the aim is to write 'beautiful' code that performs its intended purpose in an elegant way. This source of motivation may suffer if the concrete form of expression has to be compromised in order to accommodate software patents. Obligation-based factors may be positively affected because a "patent threat" may increase the sense of community. It may also increase the resolve of FOSS developers to provide non-infringing free substitutes. One example is the OGG format, a free replacement of the MP3 audio format that is argued to allow for smaller file size and higher quality.

We end this section by considering the situation in which the FOSS developer is the holder of a software patent. Table 5 summarizes the arguments. This situation is probably very rare. We are not aware of such cases except for the RTLinux patent debate in 2001. As in Table 4, motivational factors that are most likely to remain unaffected by patents are not listed.

Table 5: Potential effects of 'holding software' patents on extrinsic and intrinsic motivations

Motivation factor	**Positive effects**	**Negative effects**
Extrinsic (instant):		
Earn money/income (monetary)	(+) Royalties provide additional income.	(-) Cost of defending patent reduces income.
Learn skills (disclosure)	(+) Patent pursuit focuses search for a new solution = learning.	none
Extrinsic (delayed):		
Future career (monetary)	(+) Developer perceived as innovative is more attractive for employers.	none
Intrinsic (enjoyment-based):		
Artistic self-expression	(+) Owning a patent generates self-affirmation	none
Solve difficult problems (ego-boosting)	(+) Owning a patent generates self-affirmation	none
Intrinsic (obligation-based):		
Observance of community norms	none	(-) Community sanctions "traitors"

By-and-large, most of the potential effects of patents on extrinsic and intrinsic motivations to participate in FOSS development, and therefore also on innovation, appear to be negative. Yet, empirical research will have to show whether these arguments do in fact lend support to the claims of software patent critics. This research will have to rely on surveys administered to random samples of FOSS developers from different jurisdictions (with variation in terms of stronger and weaker software patent availability) and different domains of FOSS activity (with variation in terms of "patent exposure").

5. Conclusion

FOSS has in recent years experienced a strong expansion while, at the same time, public and expert debates on the desirability of patents on software have intensified. These two phenomena have been connected in that critics of software patents have claimed that such patents have negative spill-over effects for FOSS development. Proprietary software and FOSS development rely on very different innovation models. The former relies on very restrictive copyright practices and, depending on the jurisdiction, also on patenting functional features of software. The FOSS community uses a permissive form of copyright protection, which is designed mainly to prevent private appropriation of FOSS, and patents are usually an anathema. Whether software patents are conducive to innovation in the proprietary realm remains contested and the very few empirical studies that exist are inconclusive. Whether software patents have any effect on innovation in the FOSS realm is almost completely open. Patents could, in principle, also be obtained on FOSS, but developers (self-) selected into the FOSS community are, by-and-large, either not interested in or openly hostile to patenting of software. Therefore, it is likely that patents that are sought and/or granted on proprietary software have either no or a negative effect on FOSS innovation.

We outlined three potential research strategies for studying the effects of software patents on FOSS innovation: identification of whether FOSS developers have in fact become targets of litigation over infringements on software patents; analysis of whether, controlling for other determinants, FOSS innovation in areas where patenting in the related proprietary software area is “thicker” is slower or smaller; analysis of whether FOSS developers’ participation is negatively affected by patents.

We argued that the third strategy is likely to produce the most interesting results. To that end we presented a model that illuminates the effects of patents on extrinsic and intrinsic motivations of FOSS developers, assuming that strong motivations are a necessary condition for innovation. We hope that this model can serve as a starting point for a concerted effort to investigate whether software patents have any effect on the FOSS community and, if so, whether this effect is negative, how it operates, and what could be done to avoid negative side-effects of software patenting.

References

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

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

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

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

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

Benkler, Y. (2002). Coase's Penguin, or Linux and the Nature of the Firm. *Yale Law Journal*, (112)369, 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. (2007). An Empirical Look at Software Patents. *Journal of Economics & Management Strategy*, (16)1, 157-189.

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

Bessen, J., Maskin, E. (2000). *Sequential Innovation, Patents, and Imitation*. MIT Dept. of Economics, Working Paper No. 00-01. http://www.researchoninnovation.org/patrev.pdf

Bessen, J., Meurer, M.J. (2008). *Patent Failure - How Judges, Bureaucrats, and Lawyers Put Innovators as Risk*. Princeton: Princeton University Press.

Bitzer, J., Schrettl, W., Schröder, P.J.H. (2004). *Intrinsic Motivation in Open Source Software Development*. Free University 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, 1243–1258.

Campbell-Kelly, M., Valduriez, P. (2005). *An Empirical Study of the Patent Prospect Theory: An Evaluation of Antispam Patents*. SSRN, Working Paper .

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

Cohen, J.E., Lemley, M.A. (2001). Patent Scope and Innovation in the Software Industry. *California Law Review*, (89)1, 1-57.

Cohen, W.M., Nelson, R.R., Walsh, J.P. (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.

Dalle, J., David, P.A. (2005). Allocation of Software Development Resources in Open Source Production Mode. In Feller, J. (Ed.), *Perspectives on free and open source software* (pp. 297-328). Cambridge, MA: MIT Press.

Deci, E.L., Ryan, R.M. (1985). *Intrinsic Motivation and Self-Determination in Human Behavior*. New York: Springer.

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

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

Earnshaw, N.C. (2004). *The Samba Project: Transformation of Self through Open Source Software Development (Honour's thesis)*. Retrieved 21.06.2006 from http://samba.org/samba/news/articles/earnshaw_thesis.pdf.

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

Fagerberg, J. (2005). Innovation - A Guide to the Literature. In Fagerberg, J., Mowery, D.C., Nelson, R.R. (Eds.), *The Oxford Handbook of Innovation* (pp. 1-26). Oxford: Oxford University Press.

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), 1-26.

Free Software Foundation (2008). *Examples of Software Patents that hurt Free Software*. Retrieved 18.09.2008 from http://www.gnu.org/patent-examp/patent-examples.html.

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

Ghosh, R.A. (1998). Cooking Pot Markets: An Economic Model for the Trade in Free Goods and Services on the Internet. *First Monday*, (3)3, no pagination.

Ghosh, R.A., 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 www.infonomics.nl/FLOSS/report.

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

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

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

Haunss, S., Kohlmorgen, L. (n.d.). Political claims-making in IP conflicts. In Haunss, S., Shadlen, K.C. (Eds.), *The Politics of Intellectual Property* (p. n. pag.). Cheltenham: Edward Elgar Publishing (forthcoming).

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), 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, 1159–1177.

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

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

Hoppen, N., Beimborn, D., König, W. (2003). The impact of software patents on the structure of the software market. *Proceedings of the Eleventh European Conference on Information Systems*. Naples, Italy.

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), 531-557.

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

Klang, M. (2004). Free software and open source: The freedom debate and its consequences. *First Monday*, (10)3, no pagination.

Klemens, B. (2005). *Math you can't use*. Washington, DC: Brookings Institution Press.

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

Kollock, P. (1999). The Economies of Online Cooperation: Gifts and Public Goods in Cyberspace. In Smith, M., Kollock, P. (Eds.), *Communities in Cyberspace* (pp. 200-269). London: Routledge.

Krishnamurthy, S. (2006). On the intrinsic and extrinsic motivation of FLOSS developers. *Knowledge, Technology, & Policy*, (18)4, 17-39.

Krishnamurthy, S. (2005). An Analysis of Open Source Business Models. In Feller, J. (Ed.), *Perspectives on free and open source software* (pp. 279-296). Cambridge, MA: MIT Press.

Lakhani, K.R., Wolf, R.C. (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* (pp. 3-21). Cambridge, MA: MIT Press.

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

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

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

Levine, L., Saunders, K. (2004). Software Patents: Innovation or Litigation? In Fitzgerald B., Wynn, E. (Eds.), *IT Innovation for Adaptability and Competitiveness, IFIP 8.6 Working Conference on IT Innovation for Adaptability and Competitiveness* (pp. 229-242). Leixlip, Ireland: IFIP.

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

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

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), 10-26.

Mann, R. (2004). *The Myth of the Software Patent Thicket*. University of Texas School of Law, Working Paper No. 44. http://law.bepress.com/alea/14th/art44/

Marinova, D., Phillimore, J. (2003). Models of Innovation. In Shavinina, L.V. (Ed.), *The International Handbook on Innovation* (pp. 44-53). Oxford: Elsevier.

Mazzoleni, R., Nelson, R.R. (2004). Economic Theories about the Benefits and Costs of Patents. In Maskus, K.E. (Ed.), *The WTO, intellectual property rights and the knowledge economy* (pp. 148-169). Cheltenham: Edward Elgar.

Menell, P.S. (2000). Intellectual Property: General Theories. In Bouckaert, B., De Geest, G. (Eds.), *Encyclopedia of Law and Economics* (pp. 129-187). Cheltenham: Edward Elgar.

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

Mitchell, H.C. (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* (pp. 393-446). Cambridge, MA: MIT Press.

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

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

OSOR (2008). *Open Source Observatory and Repository for European public administrations* . Retrieved 05.09.2008 from http://www.osor.eu.

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

Quah, D.T. (2002). *Digital Goods and the New Economy*. London School of Economics, Working Paper No. 3846. cep.lse.ac.uk/pubs/download/dp0563.pdf

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

Roberts, J.A., Hann, I., Slaughter, S.A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, (52), 984-999.

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

Rossi, C., Bonaccorsi, A. (2005). *Intrinsic vs. extrinsic incentives in profit–oriented firms supplying Open Source products and services*. First Monday, (10)5, no pagination.

Ryan, R.M., Deci, E.L. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, (25), 54–67.

Sakakibara, M., Branstetter, L. (2004). Do stronger patents induce more innovation? Evidence from the 1988 Japanese patent law reform. In Maskus, K.E. (Ed.), *The WTO, intellectual property rights and the knowledge economy* (pp. 544-567). Cheltenham: Edward Elgar.

Sattler, H. (2003). Appropriability of product innovations: an empirical analysis for Germany.

International Journal of Technology Management, (26)5/6, 502-516.

Saviotti, P.P. (1997). Innovation systems and evolutionary theories. In Edquist, C. (Ed.), *Systems of innovation: technologies, institutions and organizations* (pp. 180-199). London: Pinter.

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, 29-41.

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

Stallman, R.M. (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?* (pp. 429-459). Brussels: 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, 1-39.

von Hippel, E. (2005). Open Source Software Projects as User Innovation Networks. In Feller, J. (Ed.), *Perspectives on free and open source software* (pp. 267-278). MA: MIT Press.

von Hippel, E., von Krogh, G. (2003a). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, (14)2, 209-223.

von Hippel, E., von Krogh, G. (2003b). Special issue on open source software development. *Research Policy*, (32)7, 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, 1217–1241.

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

Wheeler, D.A. (2007). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!*. Retrieved 05.09.2008 from http://www.dwheeler.com/oss_fs_why.html.

Wheeler, D.A. (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*. Cambridge, MA: MIT Press.

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

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