DISS. ETH NO. 18544

THE EFFECTS OF SOFTWARE PATENT POLICY ON THE MOTIVATION AND INNOVATION OF FREE AND OPEN SOURCE SOFTWARE DEVELOPERS

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

MARCUS MANFRED DAPP

Dipl. Betriebs- und Produktionsingenieur ETH

13.04.1974

citizen of Germany

accepted on the recommendation of

Prof. Dr. Thomas Bernauer, examiner
Prof. Dr. Stefan Bechtold, co-examiner
Prof. Dr. Georg von Krogh, co-examiner

2009

To my parents, Erika and Manfred. Thank you for everything.

Acknowledgements

Writing a dissertation may sometimes be a lonely act, but it is not something done alone. There are many people who helped me in various ways throughout this journey and to whom I owe much gratitude.

First of all, my supervisor Thomas Bernauer, who provided me with the proverbial academic freedom, allowed me to spend the time it takes to explore a new field and who also supported less conventional doctoral student' activities such as an own lecture that allowed me to see beyond free/open source software and really explore the whole field of 'digital sustainability'. Further, I thank Georg von Krogh for sharing his enthusiasm for high quality research and for giving me occasional asylum in his research group. I also thank Stefan Bechtold for his invaluable input in legal matters and thorough feedback. And of course, the group of anonymous referees and ETH Zurich who provided funding (ETH Research Grant TH -2/05-2). Thank you for making this research project possible.

Next I owe a big thanks to my friend Daniel Kammerer, long-term office mate and steady moral supporter through the highs and lows of our joint academic journey. We made it! I also thank former and current members of Thomas Bernauer's research group: Vally Koubi for teaching me how to defend my work; Stéphanie Engels, Jazmin Sejas, Stefanie Walter, and Thomas Sattler for paving the way and letting me learn by example; Anna Kalbhenn, Gabi Ruoff, and Lena Schaffer for showing me how much I don't know about statistics; Sophie Perrin, Lucas Beck and Jürg Vollenweider for moral support and the funny breaks in and outside the office. Good luck with your projects! Last but not least, I thank Claudia Jenny for being the perfect secretary and patient proofreader of my papers.

In the local academic neighborhood, I want to thank my friend Matthias Stürmer, peer doctoral student and busy FOSS advocate, whose surname is well-deserved. Further, my friends and colleagues at the Chair of Strategic Management and Innovation at ETH Zurich, especially the fantastic group of post-docs consisting of Stefan Häfliger, Sebastian Späth, and Martin Wallin. Thank you so much for your ongoing support and valuable feedback! In the wider academic world, I thank Reto Hilty, Director of the Max Planck Institute for Intellectual Property, Competition and

Tax Law for hosting me in Munich, as well as Luigi Innocente and Christophe Geiger for showing me that nothing is really easy in intellectual property law. Further, I thank Megan Conklin and James Howison for running the FLOSSmole project as well as Bob English and Charles Schweik for sharing their project classification data. You all helped me to produce a better survey.

I thank the literally thousands of FOSS developers who suffered through my survey questionnaire instead of producing excellent software. I will not interrupt you again, promised! Thanks also go to the survey lottery sponsors: Chris DiBona of the open source program office of Google Inc., Michael 'Mickey' Lauer of Openmoko Inc., and Immo Noack of ETH Zurich's Neptun program. Thanks for sponsoring a FOSS developer's dream collection of cool and hackable gadgets.

In my private life, a couple of individuals had to cope with me during most of the dissertation years and showed surprisingly much understanding in the process: my good friend Cornelia, my brother Thomas, who finally uses Linux, and my Zurich flat-mate Christian, who still does not. Thanks for your continuous encouragement, moral support, and understanding of 'why it is taking so long'.

Finally, I thank my parents, Erika and Manfred, for making it possible for me to stay in 'school' so many years before I finally got a 'real job'. This work is dedicated in its entirety to you.

All programs used for this dissertation are free/open source software, except one: you know who you R!

Table of contents

Summary

Despite ongoing debate about the effects of software patents on software development in general, no dedicated empirical studies exist that systematically investigate the claims raised by proponents and opponents of software patenting in the context of Free/Open Source software (FOSS). The question carries additional weight for two reasons: one, because the unorthodox innovation model of FOSS puts into question some of the assumptions of traditional intellectual property theories; two, because the empirical support for the general innovation-fostering effects of patenting is far from being overwhelming – particularly in the proprietary software industry, where a few empirical studies exist.

In this dissertation, I contribute to the understanding of how software patents affect the innovation behavior of developers of Free/Open Source software by introducing a novel theoretical framework that give the motivational setup of FOSS developers a central role. I also present two empirical analyses that put the individual developer in the foreground. Methodologically, new metrics for individual innovation behavior and software patent pressure are constructed and – together with the hypotheses derived from the theoretical framework – tested on a newly created set of survey data.

The theoretical framework analyses in detail how the sixteen motivational factors collected from the FOSS motivation literature may be affected by the presence of software patents. As a results from the theoretical analysis, I expect software patents to increase the levels of extrinsic motivation and decrease the level of intrinsic motivation.

In the first empirical study, the effects of software patent presence on selected, representative motivational factors is investigated. To do that a concept of software patent presence is introduced that not only includes jurisdictional components (legal availability and patent incidents), but also a new metric that captures the varying patent pressure across software domains. Empirically, none of the two camps of proponents and opponents of software patents find support: software patents do not negatively affect intrinsic motivation as predicted by opponents, but they also do not positively affect extrinsic motivational factors as proponents predict.

In the second empirical study, the analysis is extended. Here, the effects of software patent presence and motivational setup on the individual innovation behavior of FOSS developers are investigated. To do that a new metric for innovation behavior is introduced that ranks individual code contributions according to their level of innovativeness by distinguishing algorithm-based from reuse-based contributions. The argument is that algorithm-based contributions are more innovative than reuse-based contributions. Beside this ordinal innovation scale, 'reverse-engineering' as a specific type of code contribution that is important to the FOSS world, is also analyzed.

One key result is that intrinsic motivation triggers more innovative code contribution, while extrinsic motivation correlates with less innovative code contributions. The presence of software patents (using all three metrics mentioned earlier) does not seem to play a significant role, except in the case of reverse-engineering. This type of code contribution correlates with a stronger presence of software patents.

Zusammenfassung

Obwohl die Debatte um die Auswirkungen von Softwarepatenten auf die Softwareentwicklung im allgemeinen und Freie/Open Source Software (FOSS) im besonderen weiter geführt wird, existieren keine dedizierten Studien, welche die Behauptungen der Befürworter und Gegner von Softwarepatenten systematisch und empirisch untersuchen. Die Fragestellung trägt aus zwei Gründen Gewicht: Erstens, weil das unorthodoxe Innovationsmodell von FOSS einige Annahmen traditioneller Theorien Geistigen Eigentums in Frage stellt; Zweitens, weil die empirische Unterstützung für eine generelle innovationsfördernde Wirkung von Patentierung bei weitem nicht überwältigend ist – insbesondere in der proprietären Softwareentwicklung, wozu es einige wenige empirische Studien gibt.

In dieser Dissertation erweitere ich unser Verständnis davon, wie Softwarepatente das Innovationsverhalten von Entwicklern Freier/Open Source Software beeinflussen, indem ich einen neue theoretischen Erklärungsrahmen einführe, in welchem das Motivationsgefüge der FOSS-Entwickler eine zentrale Rolle einnimmt. Ebenso präsentiere ich zwei empirische Analysen, die den einzelnen Entwickler in den Vordergrund stellen. Aus methodischer Sicht werden neue Metriken für das individuelle Innovationsverhalten und den von Softwarepatenten ausgehenden Druck konstruiert, die – zusammen mit den aus dem Theorieteil abgeleiteten Hypothesen – mit Hilfe eines in einer Online-Umfrage neu erhobenen Datensatzes getestet werden.

Im theoretischen Teil analysiere ich detailliert, wie die sechszehn verschiedenen – aus der Motivationsliteratur der FOSS-Forschung zusammengetragenen – Motivationsfaktoren auf die Präsenz von Softwarepatenten reagieren können. Als Ergebnis dieser Analyse ist zu erwarten, dass Softwarepatente eine verstärkende Wirkung auf extrinsische und eine abschwächende Wirkung auf intrinsische Motivationsfaktoren haben.

In der ersten empirischen Studie werden die Effekte von Softwarepatent-Präsenz auf ausgewählte, repräsentative Motivationsfaktoren untersucht. Dazu wird das Konstrukt ‚Softwarepatent-Präsenz' eingeführt, welches nicht nur jurisdiktionale Elemente (rechtliche Verfügbarkeit und Patentzwischenfälle) umfasst, sondern auch eine neue Metrik, die den unterschiedlichen Patentdruck in verschiedenen Software-Bereichen berücksichtigt.

Keines der beiden Lager von Befürwortern und Gegnern findet empirische Unterstützung: Weder beeinflussen Softwarepatente intrinsische Motivation negativ, wie von den Gegnern vorausgesagt; noch beeinflussen sie extrinsische Motivation positiv, wie von Befürwortern vorausgesagt.

In der zweiten empirischen Studie wird die Analyse erweitert; nun werden die Auswirkungen der Präsenz von Softwarepatenten auf das Motivationsgefüge und das individuelle Innovationsverhalten der FOSS-Entwickler untersucht. Dazu wird eine neue Metrik eingeführt, welche individuelle Code-Beiträge gemäss ihrer Innovationskraft ordnet, indem – grob gesprochen – zwischen algorithmen- und wiederverwendungs-basierten Code-Beiträgen unterschieden wird. Die Ausgangsüberlegung ist, dass algorithmen-basierte Beiträge – das Neuformulieren von Algorithmen in einer Programmiersprache – einen stärkeren Innovationsgehalt besitzen als Beiträge, die auf der Wiederverwendung von Softwarecode beruhen. Ausserhalb dieser Ordinalskala wird das ‚reverse-engineering' als eigenständiger Typus von Code-Beiträgen analysiert, weil es eine wichtige Rolle in der FOSS-Welt spielt.

Ein zentrales Ergebnis ist, dass intrinsische Motivation tendentiell innovativere Code-Beiträge auslöst, während extrinsische Motivation mit weniger innovativen Code-Beiträgen korreliert. Die Präsenz von Softwarepatenten (gemessen anhand aller drei erwähnten Metriken) scheint, ausser im Falle des Reverse-Engineering, keine signifikante Rolle zu spielen: Dieser Typus an Code-Beiträgen korreliert jedoch mit einer stärkeren Präsenz von Softwarepatenten.

List of Abbreviations

AL1	Algorithm-based code contribution (coding known algorithms)
AL2	Algorithm-based code contribution (creating new algorithms)
COM	Communication (software category on SourceForge)
CVS	'Concurrent Versioning System' (tool for software development)
DBS	Database (software category on SourceForge)
DKT	Desktop (software category on SourceForge)
EDU	Education (software category on SourceForge)
FMP	Frameworks and protocols (software category on SourceForge)
GME	Games and entertainment (software category on SourceForge)
EDT	Editors (software category on SourceForge)
EPC	European Patent Convention
EPO	European Patent Office (Munich)
FOSS	Free/Open Source software
GNU	'GNU is Not Unix' (FOSS project for a free replacement of UNIX)
IP	'Intellectual Property' (umbrella term for copyright, patents, etc.)
LIB	Reuse-based code contribution (linking libraries), variable
MIM	Multimedia (software category on SourceForge)
NET	Network (software category on SourceForge)
ODF	'Open Document Format' (open ISO standard for document formats)
OFB	Office/Business (software category on SourceForge)
OOXML	'Office Open XML' (XML-based document standard by Microsoft)
R&D	Research and Development
REV	Reverse-engineering-based code contribution
ROW	'Rest of world'
RU1	Reuse-based code contribution (with little code adaption)
RU2	Reuse-based code contribution (with much code adaption)
SCE	Scientific/Engineering (a software category on SourceForge)
SEC	Security (software category on SourceForge)
SF	'SourceForge.net' (popular FOSS portal)
SWD	Software development tools (software category on SourceForge)
SWP	Software patent

SYS	System software (software category on SourceForge)
TRIPS	Trade-Related Aspects of Intellectual Property (annex to WTO agreement)
USPTO	United States Patent and Trademark Office
WIPO	World Intellectual Property Organization (a UN agency)
WTO	World Trade Organization

Chapter I

Introduction

Abstract

Chapter I introduces the topic of the dissertation and presents the key findings, derived policy recommendations, and some thoughts on limitations and future research. The first section introduces the reader unfamiliar with the research field to the general debate of innovation in the digital age. It starts with a brief history of the legal protection of software that also describes the basics about copyright and patent law. Then a more detailed presentation of the debate between the 'individual innovation' camp and the 'collective innovation' camp is given using the example of software patents and Free/Open Source software (FOSS), respectively. The two main research questions addressed in this dissertation conclude the introduction.

The second section presents a summary of the findings. Theoretical findings are rooted in the legal and motivational differences of the FOSS innovation model compared to the proprietary software model. Under empirical findings the new metrics proposed to measure domain-specific software patent pressure and innovation behavior at the individual level are presented first. They are followed by the more substantive results concerning the correlations between motivational setup and innovation behavior as well as software patent presence and innovation behavior. Methodological findings present in some detail the choice of the unit of analysis, how the two new metrics are motivated and constructed, and, finally, how data collection was organized to ensure high data quality.

Based on the findings, the third section offers a few policy recommendations. One key advice is to consider the correlation found between intrinsic motivation and higher levels of innovation behavior compared to extrinsic motivation and lower levels of innovation behavior in future policy decisions. FOSS is not just another 'industry sector' where patents as instruments function just as expected. In addition, the characteristics of FOSS (public-good-like availability of software) give it a potential for social benefit that the proprietary software sector cannot offer.

The final section briefly discusses some limitations of the dissertation and related ideas for future research. The main challenges were the measurement of software patent presence in a non-trivial yet simple way and the sampling bias introduced by focusing on 'alive' FOSS projects for the survey. Derived from these challenges are the ideas for future research: to continue working on metrics for software patent presence and dedicate studies to 'dead' projects to identify causes of death; and, finally, to figure out why FOSS developers do reverse-engineering!

1. Innovation and the digital age

At first glance, the research questions investigated in this dissertation appear to cover a rather narrow field, of interest only to software experts, legal scholars, innovation economists, and digital freedom activists – and maybe not even all of them. Yet, the arguments raised and results discussed here are only examples in a much wider debate about the sources of innovation and originality in works of the human mind and their adequate level of legal protection.

These questions were answered a few hundred years ago; with 'copyright', 'patents', and other types of 'intellectual property' (Menell 2000). The digital age, however, requires that we address these questions anew. Over the last decades, we have all experienced, how personal computers have dramatically accelerated the speed with which we can create and modify works of the mind. But we have only just begun to see the fundamental change and potential, the Internet is bringing to individual innovation processes. On the downside, we are debating software patents and longer copyright for scientific and other literary works as well as unauthorized sharing of music and movies; on the plus side we are contemplating the potential of free/open source software, the open access paradigm to scientific knowledge, or Wikipedia, openstreetmap, and many other non-commercial collective efforts to produce useful knowledge that were impossible before networked digital technology (Benkler 2006). The research in this dissertation connects two of the examples above. Software patents, the symbol for the traditional understanding of innovation as the individual genius inventor who needs protection, are on one side; Free/Open Source software (FOSS), showing how innovation can also happen by applying legal protection instruments in any but the traditional way, are on the other side.

This introductory section provides a contextual background to the research presented in this dissertation, because its interdisciplinary nature brings technical and legal complexity with it. It starts by briefly summarizing the historical origins of software protection through copyright and patents. Then, the patenting development is contrasted with the emergence of free/open source software as the two symbols for individual and collective paradigms of innovation. The section concludes by formulating the research questions addressed in this dissertation. The following section highlights the main theoretical and empirical findings of the dissertation. Policy recommendations are suggested in the section thereafter. The final section addresses some of the limitations encountered during the whole research project and also provides ideas for future research.

1.1 The legal protection of software

Does innovation only require an individual genius or is a larger group of humans needed to come up with something new? The question of whether innovation is an individual or a collective process is not only an academic, but also a very practical one. The traditional systems of patents and copyright are rooted in the individual paradigm (Lévêque and Ménière 2004) and have difficulties in accommodating new forms of collective innovation and authorship. However, networked digital technology enables exactly such collective innovation with FOSS being the pioneering example (Lessig 2002; Benkler 2006; see also Suber (1998) for a philosophical introduction to the concept of software).

Why are patents (and copyright) designed the way they are? The reoccurring key argument is that the creation of new knowledge is costly and can very easily be appropriated by a free-rider.[1] To protect the inventor (or author in copyright terms), he[2] is handed an exclusive, time-bound right excluding everybody else from the commercial exploitation of his invention (or work). The only way to access the invention or work is by paying for a license (lat. 'permission'). Today, we have license fees for TV shows, music performances, sports training programs, images, videos, and all sorts of technical inventions.

Despite this common basis, there are two fundamental differences between copyright and patents, which are explained briefly as they both are relevant to the discussion of software. Copyright protects the form/expression of an idea or concept. Einstein's books on the theory of relativity are copyrighted, but the theory itself (the idea) is not. Therefore, anyone could write a book on the theory of relativity and compete on the book-shelf. Patents work in the exact opposite way: they protect the underlying idea/concept and thereby all possible forms covered by this idea. Therefore, if I patent a vehicle with two wheels and a chain propulsion mechanism powered by a human, I have patented a bicycle. But if I leave the number of wheels (or humans!) unspecified, my patent also covers tricycles, tandems, etc. The scope of a patent is defined by the patent claims it contains and usually the patent applicant is interested in obtaining broad patent protection.

1 Easy copying is usually considered a problem, especially in 'developed' knowledge economies. Opinions, however, may change over time. Gerster (2001:6) provides the interesting historical example of Switzerland's chemical industry in the 19th century, whose success stemmed from not having a patent law and hence no legal basis for patents that rivaling Germany could use. Switzerland was accused of being a 'pirate state'. Yet, the industry prevented every attempt of legislating a patent law – quite contrary to the standpoint of today's Swiss chemical industry, as Gerster points out (Ibid:22). Boldrin and Levine (2008, chapter 8) give a more systematic analysis of Switzerland's case.

2 Throughout this dissertation "he" refers to both genders.

The second difference between copyright and patents can be derived from the first difference. Patents require an explicit registering and granting process, during which the application has to pass a 'patentability test'. Patents costs money to obtain. In contrast, copyright is implicit and you get it for free the moment you create an original work, such as a dissertation.

It is interesting to see how the question of legal protection has been answered differently at various times in the young history of software (Grassmuck 2004, chapter "Geschichte der Software-Entwicklung"). In the first phase (1960s), the computer market consisted of huge mainframes that were sold with software included as an "instruction manual". Most of the software needed was written by the users themselves – and widely shared. Software was not considered a product by itself and no separate software market existed. In the early 1970s, the US Department of Justice brought pressure to break up IBM because of its monopolistic position. To avoid the break-up, IBM decided to unbundle its hardware- and software-business, therewith creating the nucleus of today's software industry. During that time, many of the large and now well-known software companies were created (Microsoft 1975; Apple 1976; Oracle 1977). It is remarkable to note that the software industry got established without software having any legal protection for several more years.

The second phase started with the gradual adoption of the copyright system for software in more and more countries.[3] Yet, this decision was neither obvious nor the result of a general international understanding of the matter. In fact, in the late 1970s, the World Intellectual Property Organization (WIPO) was working on a new so-called sui generis (lat. 'of its own kind') protection mechanism explicitly designed to accommodate the specific characteristics of software because copyright and patents were considered unsuitable for the protection of software. The US, however, created a fait accompli in 1980 by deciding that software should fall under copyright law. Other countries were persuaded by the US to also adopt copyright as the protection mechanism of choice (Hilty and Geiger 2005). The following two decades the software industry skyrocketed, frequently creating monopolistic/oligopolistic market constellations that gave rise to anti-trust law suits. (A discussion why software may have an inherent propensity to create monopolistic market structures – and how copyright and patents may reinforce that – can be found in the first chapter.)

3 See Scotchmer (2004) for a general, yet compact introduction to the political economy of the international intellectual property (IP) system, including a brief history of the international IP system.

1.2 The software patent debate and the 'promise' of FOSS

A third phase in software history can be identified with the rise of patent protection for software. The origins lie in the US, where several court decisions beginning in the 1980s started to dilute the hitherto self-understood exclusion of software from patentability (Jaffe 2000), hence called the 'software patent experiment' by Bessen and Hunt (2004). Consequently, software in the US is copyrighted and can be patented at the same time, resulting in complex legal analyses and ever more law suits (ibid.). The EU was following close on the heels of the US with a directive proposal on the 'patentability of computer-implemented inventions' that the European Commission introduced in 2002. After a long and heavy debate with strong lobbying from software patent proponents and opponents, the proposal was finally rejected in second reading by the EU parliament in 2005. Thus, software in the EU cannot be patented de jure, but de facto several thousand patents have been granted that qualify as software patents.

In a fourth phase, the 1980s saw a renaissance of the original idea of unprotected software at just the time the "proprietary" software industry, whose business model fundamentally relies on copyright and trade secrets (keeping the modifiable source code secret), was quickly expanding. The 'free software movement' has been promoting 'software freedom' ever since, meaning that every user should have the freedom to run, modify, copy, and distribute software as he pleases (Free Software Foundation 2006; see Stallman (1984) for the philosophical motivation of the 'GNU project' that was aimed at building a free replacement to the UNIX operating system and constitutes a big part of most Linux versions today). Interestingly, free software is made 'free' by applying the same copyright mechanism, but for a whole different purpose: instead of exclusion, the typical aim of copyright, the aim is inclusion by enabling everyone to participate.

A technical precondition for modifications is that the program is available in its source code form. In the early 1990s, 'Linux' was created and quickly completed the GNU system, leading to a fully equipped free operating system. To promote it and to remove the political connotation of "freedom", the term "open source software" (Open Source Initiative 2006) was introduced in the late 1990s to make the idea more appealing to business. Therefore, throughout this dissertation I use the acronym FOSS, standing for 'Free/Open Source software'.

Linux came along just at the right time to take advantage of the emerging Internet in the early 1990s. The collaboration and sharing that for much of the GNU project had taken place by sending magnetic types or using telephone lines to send packages of code in a one-to-one ap-

proach, was given a boost through the possibilities offered by the Internet: Massive sharing and exchange of program code has been made possible at very low cost, on a global scale. As every developer contributes small pieces to the mosaic, every developer and even non-developers benefit from free access to large and constantly improved software systems. FOSS developers are driven by a broad spectrum of motivations (Krishnamurthy 2006) and many, though by far not all, contribute code without monetary compensation. This model is shaking the established software industry and directly undermining its business model of selling copyright licenses to allow customers to use their software.

1.3 Research questions

The dissertation addresses two developments. The first is the concept of patenting software that transfers the arguments of patent proponents to the software realm: Innovation in software (as in other fields) happens if individual genius creators have enough monetary incentive to invent. The second is the FOSS phenomenon that is characterized by collective innovation, shared ownership (using copyright) and often volunteer contributions of many developers around the world.

The dissertation is located at the intersection of these two developments and connects them theoretically by raising the following research questions:

Question 1: How does the presence of software patents affect the motivational setup of FOSS developers?

Motivational setup is one of the most investigated fields in FOSS research. As will be shown in the first chapter, the spectrum of motivational factors is broad and includes monetary and other factors. It is important to analyze which of these factors can (theoretically) be affected by software patents in order to be able to formulate hypotheses about the effect.

Question 2: How does the presence of software patents and the motivational setup together affect the innovation behavior of FOSS developers?

Software patenting is one of the least investigated fields in FOSS research. Although motivational setup is important in the FOSS system, the ultimate performance benchmark for patents is innovation. Do they affect the innovation behavior of FOSS developers and if yes, in which ways?

The questions are further developed into sets of testable hypotheses relating software patent presence and motivational setup (chapter III) and software patent presence and innovation behavior (chapter IV).

In addition, as this study is the first to address these specific questions and link theories of patenting and FOSS motivation, some new concepts and measurements had to be developed that will also be described in the next section.

2. Overview of main findings

2.1 Theoretical findings

Theoretical findings can be summarized by legal and motivational differences in the innovation models of the FOSS and the proprietary software system.

Legal differences in the innovation model. 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. Patents could, in principle, also be obtained on FOSS, but (self-)selected community developers are, by-and-large, either not interested in or openly hostile to patenting of software. This has led to the hypothesis that patents that are sought and/or granted on proprietary software have either no effect or a negative effect on FOSS development.

Motivational differences in the innovation model. Particularly the broad spectrum of motivational factors has made the theoretical analysis of software patent effects a tedious undertaking. The first model presented in chapter II specified potential effects of patents on extrinsic and intrinsic motivations of FOSS developers on a still abstract level that was further detailed for the empirical studies. The argument chains for both claims, that of FOSS advocates that opportunities for software patenting have negative effects on FOSS and that of software patent proponents that patents will help foster innovation in the FOSS field, too, have been difficult to construct. One result, however, is that traditional utilitarian patent theories that are based on incentives, rewards, and disclosure seem to trigger primarily extrinsic motivational factors – not unexpectedly, as patents are designed with extrinsic motivation in mind. Yet, these extrinsic factors are of less importance to FOSS developers.

As can be seen below, empirical findings also suggest that intrinsic motivation affects the innovation behavior of FOSS developers in a different way compared to extrinsic motivation.

2.2 Empirical findings

The empirical findings are divided into three groups: first, results concerning software patent presence; second, results connecting software patent presence and motivation setup; third, results connecting software patent presence, motivational setup, and innovation behavior. The section concludes by interpreting the results in a comparative way.

Software patent presence. Although the general approach of applying a broad concept of software patent presence and using different metrics is still useful, most of the metrics used have shown little predictive power. One clear result, however, is that software patent incidents are a very rare event: Only 2.7% of the respondents reported such an incident.

The newly constructed patent pressure index (used in chapter III and IV) proved useful, as it clearly shows the differences in perceived patent pressure between different software domains. The fourteen categories of SourceForge that were also used in the survey (see the list of abbreviations for all explanations) indicate three groupings: few domains such as multimedia (MIM) and formats and protocols (FMP) show a high patent pressure, and few domains such as education (EDU) and text editors (EDT) show nearly no pressure, which leaves a rather large group in the middle with only slight variation in the patent pressure level.

Motivational setup. Software patents do not appear to show a strong effect on FOSS developer motivation in general. This is true for both camps in the software patent debate: the presence of software patents has no positive effects on monetary and skills-related motivation, as argued by proponents; it also does not show negative effects on joy- and self-expression-related motivation, as argued by opponents. In contrast and counter-intuitively, joy-related motivation seems to be positively influenced by the presence of software patents.

Innovation behavior. Software patent presence per se has no observable empirical effect on innovation behavior, be it positive or negative, with the exception of reverse-engineering, which seems to be more frequent when software patents are present. One possible explanation is that FOSS developers who are motivated by the philosophical idea of 'free software' choose fields in which software patents are strongly present – exactly to provide free alternatives that are also 'freed' from patent claims.

The newly introduced innovation metric – code contribution types – allows the distinction between more (algorithm-based) and less (reuse-based) innovative code contributions. In the ab-

sence of established systemic metrics, it is proposed to focus on the individual level when measuring FOSS innovation. Outside the algorithm/reuse logic, reverse-engineering as a special contribution type that is relevant to the FOSS community has been included as well.

Concerning the effects of motivation on innovation behavior, strong support can be reported for the following result: Above-average intrinsic motivation (joy and self-expression in code-writing) increases the odds for more algorithm-based code contributions, while above-average extrinsic (monetary and skills-related) motivation seems to decrease the odds. In connection with reuse-based contributions, the opposite relationship finds moderate support as well: Above-average extrinsic motivation increases the odds for reuse-based contributions, while above-average intrinsic motivation decreases the odds. The third result relates to reverse-engineering: None of the five motivational factors included in the analysis seem to explain why FOSS developers engage in reverse-engineering activities.

Concerning the effects of software patent presence on innovation behavior, the empirical results are less conclusive. Neither SWP opponents nor proponents will find support for their positions that the presence of SWP decreases or increases the odds for innovative, algorithm-based contributions by FOSS developers. None of the three metrics used to capture SWP presence lends sufficient support to either side – be it positive or negative. Support is found, however, for the hypothesis related to reverse-engineering: stronger SWP presence attracts reverse-engineering-based contributions by FOSS developers.

2.3 Methodological findings

From a methodological point of view, several new approaches were taken: the individual instead of the project as unit of analysis; a new metric for individual innovation behavior; measuring software patent presence in new ways; and constructing a new data-set from a controlled survey.

Unit of analysis. The individual project leader was chosen as unit of analysis. The argument is that the leader of a FOSS project has the best overview of interferences caused by software patents, and because it was (or still is) a developer himself who is strongly attached to the project, he is the best source for information on motivation and software patent situation.

Measuring innovation behavior. A new metric for measuring innovation behavior on the individual level was developed and used for the empirical analysis. Based on the thought that differ-

ent levels of innovativeness should be reflected by the type of code contributions FOSS developers make, an ordinal scale was developed that captures 5 different levels of code contribution types. They are roughly separated by the idea that writing new algorithms (two levels) is more innovative than reusing existing code pieces (three levels). A sixth type of code contribution – reverse engineering – was also included in the analysis but was not part of the ordinal scale because it is very different from the other five types of contributions: reverse-engineering replicates functionality of existing programs without having access to the source code.

Measuring software patent presence. Measuring the presence of software patents proved to be rather difficult. Aside from the legal availability of software patents that depends on the jurisdiction, a second measure depending on the software domain was introduced. Anecdotal evidence shows that different software domains (e.g., multimedia or office programs) are differently affected by software patents. The 'wisdom of the crowds' approach was used to generate a new metric for this pressure – by asking the sampled respondents for their assessments. Together with a simple measure for software patent incidents, a range of three metrics was available to measure software patent presence: legal availability of software patent in a jurisdiction, incidents, and domain-specific patent pressure. Measuring software patent presence through different metrics is a worthwhile approach to follow in the future, as the construct of a dichotomous 'SWP Law' variable proved to be too coarse as a predictor. Further research is needed, however, in finding and refining jurisdictional and non-jurisdictional measures.

Data collection. For empirical testing a new data-set was created from an online survey in which 2441 project leaders of FOSS projects responded with information about their motivations, code contributions, and contacts the project had with software patents during the two-year period from August 2005 until August 2007. Special care has been taken to avoid self-selection of respondents, a typical weakness of previous FOSS developer surveys that used open invitations on mailing lists to attract participants. Instead, a sampling frame of all FOSS projects hosted on SourceForge in August 2006, which had an 'alive' status according to the metric of English and Schweik (2007), was created. A simple random sample of 11,000 was drawn and contacted in an automatic way. I expect that personalized invitations, a separate lottery drawing, and the impossibility of self-registration greatly reduced self-selection bias in that regard. Yet, only allowing alive projects to be randomly selected again introduced a sampling bias, which will be addressed in the research design sections of chapters III and IV.

3. Policy recommendations

This debate is relevant not only from an academic point of view, but also for regulators who are trying to build effective innovation systems without putting a particular innovation model at a disadvantage.

One key result is that intrinsic motivation leads to comparably higher levels of innovation behavior, while lower levels of innovation behavior need extrinsic incentives. An important finding that, if shown reliable, gives a clear direction for organizations and policy-makers who want to foster FOSS innovation. Intrinsic motivation appears to not only keep the FOSS system alive and kicking, but more of it also seems to lead to more innovative contributions. Simply put: 'Programming challenging, new stuff is fun'. On the other hand, it appears that reuse-based contributions with a lower innovation level – often needed for 'the last mile' before a program is end-user-ready, an area where FOSS suffers – can be supported by offering extrinsic incentives.

At the same time, the recommendation is to proceed carefully in this complex and unexplored field, as there is currently no empirical support for either the software patent proponents' nor the opponents' position. The results from this study suggest that US-based FOSS developers have gained no advantages from the strong presence of SWP in their country. Therefore, if the EU is planning to introduce a similar SWP policy (as it has tried in the past), it should provide compelling evidence of how the FOSS community's motivational setup – an important source for its performance – would be affected with regard to the key result mentioned above.

For policy-makers in innovation and intellectual property policy fields the challenges are (a) to decide whether FOSS deserves a special case when debating software patents because of its unique way of creating software for the common good; (b) to continue treading carefully in the field of software patents before jumping to legislation. The FOSS market has reached a size where harm cannot be considered collateral damage as it may have in the past. Although the results have not shown systematic harm to the FOSS communities, there is still no empirical support that the traditional arguments in favor of patents do hold for the FOSS system – or software in general, as some continue to argue.

4. Limitations and future research

As with most initial studies in a new field, there are some limitations to be acknowledged, which should be addressed in future research. First, measuring the software patent situation in a systematic, quantitative way was a considerable challenge. The crude measure for SWP law applied in this study has most probably obscured some geographical and/or cultural effects, leading to biased results and lowered predictive power.

Second, much effort was made to produce a sample of alive FOSS projects that would respond to a survey call. Yet, this effort in itself has led to a sample bias. A future study looking specifically at why dead FOSS projects actually died – or new ones were never born – would be an important addition to these results. One starting point is to include all the 'dead' projects omitted from this study in a new survey. The response rate may be quite low, but an additional well-designed series of case studies could still deliver a richer picture of the situation than we have today.

Third, taking the individual developer as unit of analysis ignores explanatory factors only visible on project level that can also influence innovation behavior, such as project size and organizational structure. The larger a project is, the more elaborate its organization structure becomes, the more contributors tend to specialize in their contributions – up to a point where dedicated roles may emerge. Such a division of labor is putting a systematic bias on the measurement of individual innovation behavior.

Fourth, the descriptive results about individual innovation behavior (e.g., FIGURE 1 on page 128) and the number of actual software patent incidents (e.g., TABLE 3 on page 90) indicate no clear differences between jurisdictions with regard to software patent presence. That raises the question whether there is de facto any large difference with regard to software patents? In other words, the possibility that software patent presence has no observable effect on the motivation and innovation behavior of FOSS developers is not yet completely ruled out. Further research is needed here.

Fifth, the representativity of the sample can be debated as none of the large FOSS projects was included and hence none of the projects that are backed by corporations. Both aspects affect the motivation and innovation setup of participating developers. New factors like project size, governance structure on project level, as well as firm level factors like business model, their position on software patenting would need to be considered.

For researchers, the challenges raised in this study are (a) to develop an easy-to use yet non-trivial metric to measure the presence of software patents empirically; (b) to quantify their effect on the FOSS system, helping policy-makers make better-informed decisions. For future research, it would be useful to verify some of the findings using other data sources. CVS logs have been used in the past for code contribution analysis. A method to classify 'real' code into the code contribution types proposed in this work would provide a powerful mechanism for replication of the results. In any case, the challenge to investigate the effects of software patents on innovation behavior of FOSS developers continues to be relevant because only if they foster innovation, should they be present in the FOSS system. With the US starting to rethink its software patent path and the EU still undecided on whether to follow in some way or choosing its own path, the timing may be just right for more research in this area.

What still remains opaque from a theoretical and empirical point of view is the question of why developers engage in reverse engineering. After all, it is an important activity in the FOSS world. A broader analysis of motivational factors is needed here and a dedicated study focusing on projects with a high ratio of reverse-engineering may be adequate. Furthermore, this research is highly relevant, too. The ODF-OOXML controversy is but one example of the rising issue of 'patents in (open) standards' – a natural neighbor to the 'patents in (open) software' debate.

5. References

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

Bessen, J., Hunt, R. (2004). The Software Patent Experiment. In OECD (Ed.), *Patents, Innovation and Economic Performance* (pp. 247-263). Paris: OECD Publishing.

English, R., Schweik, C.M. (2007). Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. *UPGRADE*, (IX)6, 54-59.

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

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

Hilty, R.M., Geiger, C. (2005). Patenting Software? A Judicial and Socio-Economic Analysis. *International Review of Intellectual Property and Competition Law*, (36)6, 615-754.

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

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

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

Lévěque, F., Ménière, Y. (2004). *The Economics of Patents and Copyright*. Berkeley: Berkeley Electronic Press.

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.

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

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

Suber, P. (1998). What is Software?. *Journal of Speculative Philosophy*, (2)2, 89-119.

CHAPTER II

Hot Debate about Chilling Effects: Do Software Patents Hamper FOSS Development?

Co-authored with Thomas Bernauer

Earlier versions of this paper were presented at the 'Frontiers of Regulation' CPR/CRI conference 2006 in Bath (UK), the Strategic Management and Innovation seminar at ETH Zurich, and the Oxford Internet Institute's Summer Doctoral Programe 2005 in Beijing (PRC). We thank all participants for helpful comments.

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").

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.[4] 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).[5] 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. Most projects are further characterized by decentralized self-governance instead of a strict command-and-control hierarchy. They are often driven by many volunteer developers who maintain and expand the code base – instead of employees directed and paid by a firm. Larger projects are usually more tightly organized and/or centralized and several firms employ developers to work on FOSS (we return to this issue further).

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 protec-

4 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 Trade-related Aspects of Intellectual Property Rights (TRIPS) annex to the WTO agreement or the World Intellectual Property Organization (WIPO) Copyright Treaty.

5 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.

tionist 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 organizations, and other actors are using FOSS[6] – 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; Lerner and Tirole, 2002) 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 and pressure in that model. The paper ends with suggestions for how the empirical relevance of the model and its hypotheses could be tested.

6 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).

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.

2.1 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 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 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).[7] 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 invention 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[8] 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 en-

7 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.

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

courage 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. 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).

2.2 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 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 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 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).[9]

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

9 Strong first mover advantages help explain why a few pioneering US software companies ended up dominating the software market. Low entry barriers help explain why European companies still established their own software market despite the US's advance; but also why the European market mostly consists of small- and medium-sized enterprises.

whom one can exchange data. Conversely, switching to another, non-interoperable program imposes considerable costs on customers. Interoperability thus creates 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 drops rapidly when a new, improved 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.

2.3 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 those 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 layperson, and appropriate for industrial application. Critics claim that software fails the first and second test: 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 blurs 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 par-

ticularly 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 mainly concentrate on the negative claims because this paper focuses on whether there are in fact negative effects.[10] The following arguments have been advanced against software patents (summarized from 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 that patenting requires disclosure – which fosters incremental and sequential improvement and thus increases diversity and interoperability – rests on the assumption that the disclosed information is sufficient 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. When we add Cohen and Lemley's 'magic words doctrine' argument mentioned above, it becomes questionable to what extent the disclosure argument is valid in the case of software patents.

Cumulative innovation such as in software development may be slowed down or even halted by a fundamental generational trade-off between inventors. Given to the first inventor, a patent creates obstacles for the next; given to the second, it reduces incentives for the first, who may

10 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 otherwise 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.

then not produce the first invention. The challenge is to compensate early innovators while ensuring incentives for later innovators (Scotchmer, 2004, ch. 5).

Using patents to prolong otherwise short innovation cycles may create advantages for some companies, but can negatively affect the economy as a whole. As the average duration of the patent examination process tends to be longer than the typical software time-to-market, the expected prolongation effect may in fact not be achieved. Several studies have investigated the patent examination process. Harhoff and Wagner (2006:17) for example note that the examination of a patent application usually takes several years, and around 4.3 years in the case of the European Patent Office (EPO). They attribute this rather long examination period to decision-making procedures, the complexity and volume of applications, and strategic behavior of applicants. Popp, et al. (2004:4) observe that the patent examination process takes around 2.4 years on average in the case of the United States Patent and Trademark Office (USPTO). They conclude that "[a]pplications in newer, more complex technologies such as biotechnology or computers take significantly longer than other patent applications." (Ibid.:41).[11] In comparison, the typical release cycles of FOSS projects are measured in months or a few years in extreme cases.

Several authors argue that patents tend to restrict imitation and thus reduce the potential for sequential innovation and positive network effects. Scotchmer (1991), for instance, criticizes the application of economic incentive theory to patents as being too simplistic in view of the modern economy. She argues that such application does not take into account the cumulative nature of innovation and the cumulative effects between generations of inventors. Moreover, as argued by Cohen and Lemley (2001:17-18) and Samuelson and Scotchmer (2002:1584), patents may also hamper reverse engineering of patented software because patent law does not provide for a reverse engineering right and disclosing the implementing source code is not required. Patent law does, therefore, not support potential innovators interested in learning how patented software functions. By implication these authors claim that free-riding in the form of imitation or reverse engineering, which is considered advantageous in software innovation (Levine and Saunders, 2004:7), is negatively affected by patents.

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

11 Most patent offices experience a considerable backlog. In a November 2005 press release, the USPTO mentioned a backlog of more than half a million patent applications. In May 2008, the controller of the EPO stated in an interview with the Intellectual Asset Management Journal that the USPTO will soon reach 1 million applications and the EPO will be there "within 5 years".

positive network externalities (Katz and Shapiro, 1985). The size of the network effect is determined by the degree of interoperability: programs tend to inter-operate 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 inter-operable 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 inter-operable programs. The incentive to "invent around" patented software code may diminish interoperability as well because it may not be possible to develop inter-operable, non-infringing software code. Non-inter-operable software is perceived by users as of limited utility even if it is more innovative (Palmer, 1989:302).

Critics also argue that patents on software may create legal uncertainties. They protect an idea in all possible forms based on a list of 'claims'. To judge which claims are violated – where exactly infringement happened and to what extent – is usually decided by specialized courts because the validity of a specific infringement claim may be difficult to assess. In principle, infringement decisions cannot be more clearly defined than the patent boundaries they are based on, and Bessen and Meurer (2008, ch. 9) in fact argue that the boundaries of software patents are particularly fuzzy and thus uncertain from a legal viewpoint.

To the extent several or many fuzzy patents overlap so called "patent thickets" may emerge (Shapiro, 2001; Federal Trade Commission, 2003:6). This problem is particularly relevant in the area of software, where innovative ideas can be implemented in a variety of ways. Consequently, even a single software patent could block many different solution paths (algorithms). In the short run, legal uncertainty and transaction costs may thus increase. In the long run, patent thickets may reduce the amount of knowledge that is freely accessible to future innovators. Underprovision of knowledge can result in less innovation, a situation usually called the "tragedy of the anti-commons" (Heller, 1997). Copyright law, in contrast, is widely accepted "as a means to prevent software from being copied" (Rossi, 2004:26); it protects a specific expression (source code) but not the underlying concept and functionality of a particular software.

Patents may be inefficient solutions to 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).

Patent critics claim, moreover, that 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).

While the list of potentially negative effects of software patenting that one encounters in the literature is longer than the list of potentially positive effects (the latter are used to justify patenting) the empirical evidence is surprisingly 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. Amabile (1996) defines innovation as the implementation of ideas through social, commercial or organizational activities. Because it exhibits distinct activities on all three dimensions, the FOSS innovation system itself can be considered a process innovation. 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 have no access to the source code) and in some cases also patents on functionalities expressed in the object code. In contrast, users and developers of FOSS obtain a greater set of rights, in particular the right to copy, modify and pass on source code as well as object code.[12] Patents on FOSS software are possible, depending on the jurisdiction, but are 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 (2006:4-5) 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".

12 FOSS licenses can differ. The Open Source Initiative and the Free Software Foundation offer license reviews and comparisons: http://www.fsf.org/licensing/licenses/, http://www.opensource.org/approval (retrieved 2009/03/10).

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, because traditional royalty-based producer-consumer-relationships do not exist. Consequently, a project's success can only partially be measured in commercial terms.[13] 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. Hence, producing software in a collaborative manner and giving it away for free is not incompatible with a software business: companies such as IBM, Red Hat or Canonical have built business models in line with the open source development model. Goldman and Gabriel (2005) and Krishnamurthy (2005) provide analyses of FOSS-based 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

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

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 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 because the number of participants is not limited to firms. 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 users' needs and developers' interests are coming together on a much larger scale than in a proprietary setting, thus leading to more heterogeneity (Bessen, 2005): a variety of developers see a variety of prospects to build on when confronted with a certain programming task. If no consent on the overall direction of a project can be found, a split (called 'fork') may result. Forking adds to the variety of approaches to solve a given problem, but also splits and therefore dilutes community resources.

Second, FOSS development is cumulative and an example for reproduction and inheritance in the evolutionary model: many 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, though 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 calculus 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 summary, 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 Software Patents Affect FOSS?

The principal justification for software patents is that they encourage innovation in the proprietary realm of software development. Does this imply that 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 itself? 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.[14] 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 weaker 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, controlling for all other influ-

14 See, for example, www.groklaw.net and www.chillingeffects.org (visited on 21.07.2008).

ences, 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.

4.1 Motivations for participation in FOSS projects

Several recent studies have 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 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. Note 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 in terms of enjoyment- and obligation-based factors.

TABLE 2: Intrinsic motivational factors

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[15]. 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).

4.2 Effects of software patents

How can we connect the software patent issue to 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 analytically 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.

15 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.

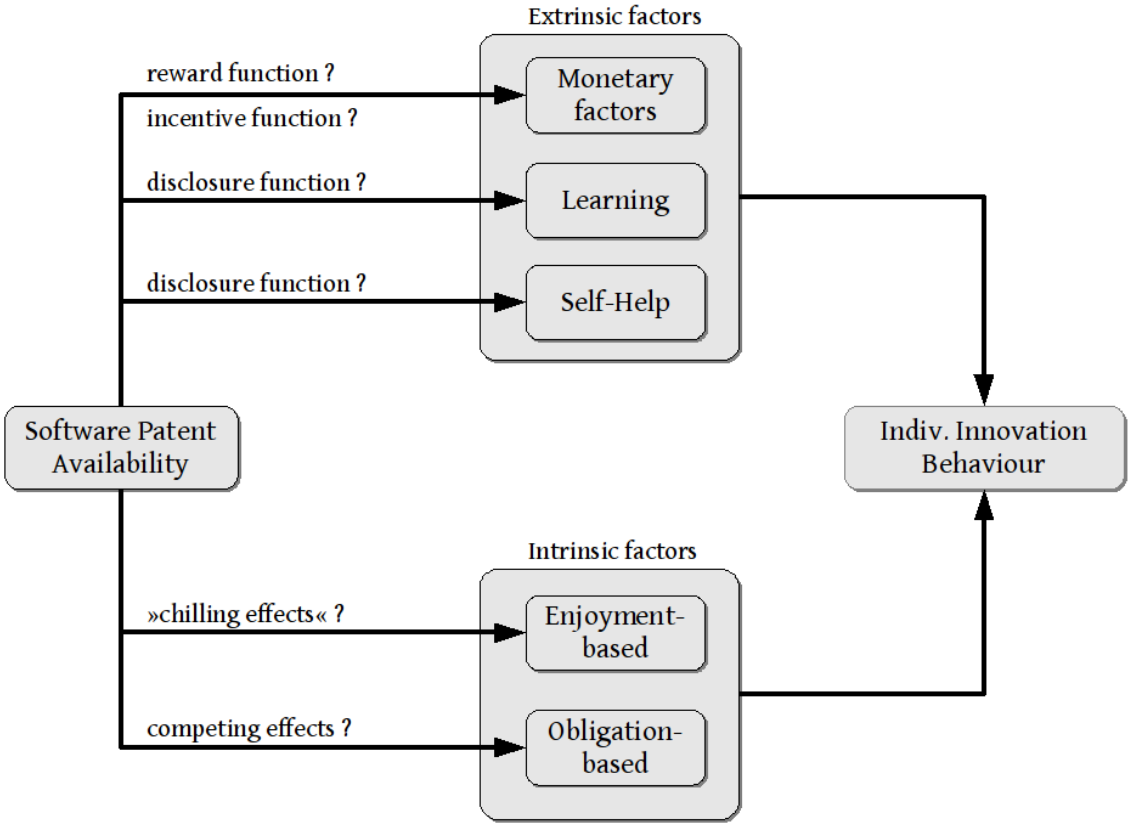

FIGURE 1: Effects of software patent availability on participation in FOSS development

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).[16]

16 Generally, software innovation is driven by two main sources: competition between 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).

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 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 adaptation. At the high end, we can place new implementations of existing algorithms or newly devised algorithms.[17] Table 3 shows 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.

As discussed above, innovation behavior is affected by two types of motivational factors (extrinsic and intrinsic). 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 and exposure can be measured both from the developer's (subjective) viewpoint and from a legal expert perspective. Both measures will have to rely on surveys of FOSS developers. Measures of the former type will generate information about the application domains (for example multimedia, security, office, etc.) in which patents appear, from the developer perspective, particularly prevalent and strong. Such surveys could also identify specific incidences where patent pressure has inhibited FOSS participation. Measures of the second type will estab-

17 Klemens (2005) raises the question of whether algorithms, which are essentially math, should be patentable subject matter at all.

lish where individual developers are located geographically and to what types of FOSS projects they contribute. This information can then be combined with assessments by legal experts of patent availability and related legal practices in specific countries and software areas. 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 specific jurisdiction and thus subject to that jurisdiction's rules and practices concerning software patenting. We can therefore examine whether patent effects differ depending on software area and 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 systematic research on this issue exists. As depicted in Figure 1 we propose to focus on three extrinsic and two intrinsic motivational factors. In the remainder of this section we discuss how patent availability might affect these factors and hence also participation in FOSS projects. Table 4 summarizes arguments for the situation where a FOSS developer faces a software patent claim emanating from the proprietary software domain.

TABLE 4: Potential effects of software patents on extrinsic and intrinsic motivations

Motivational 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 of including code
Net gain from code	none	(-) Legal risk of including 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: [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 patent pressure. 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 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.

Even though this is probably very rare[18], we should also consider the situation in which a FOSS developer holds a software patent. Table 5 summarizes the arguments. 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"

18 We are not aware of such cases except for the RTLinux patent debate in 2001.

Finally, it could be interesting to also explore whether intrinsic and extrinsic motivations affect each other and how the relationship between them is affected by patents. One possibility is that patents enhance extrinsic at the expense of intrinsic motivations and thus amplify crowding-out effects.

4.3 Empirical research strategy

By-and-large, most of the potential effects of patents on extrinsic and intrinsic motivations to participate in FOSS development, and therefore also on FOSS innovation as a whole, appear to be negative. Yet, empirical research will have to show whether these arguments do in fact support the claims of software patent critics. Such 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"). Such surveys could use the theoretical model outlined above as the basis for specific survey items.

Empirical research along these lines will have to cope with several challenges, the most important of which concerns sampling. FOSS developers use different types of web-based portals. Some of these, for example SourceForge (www.sourceforge.net), host a very large number and range of projects and involve tens of thousands of developers. Others host very specific FOSS projects – one example is the OpenOffice project (www.openoffice.org). We submit that empirical research should start with sampling from a very large and diverse FOSS portal, such as SourceForge, to obtain maximum variation on the explanatory variables and a large enough sample for meaningful statistical analysis. Technical challenges in sampling from such portals include distinguishing idle or "dead" from "alive" projects, and obtaining access to developers' email addresses in order to draw a proper random sample – simply posting a survey on a FOSS portal and waiting for self-selected developers to respond would produce unacceptable sample bias. The approach sketched here may of course result in some sample bias as well because it excludes specific FOSS projects (such as OpenOffice or Linux) as well as FOSS projects based at individual companies. Follow-up research will thus have to compare results obtained for samples drawn from different FOSS portals. Another very tricky sampling issue to be tackled is how to deal with the possibility that patent exposure has already driven the more risk adverse developers into low exposure FOSS projects or out of FOSS altogether. In the extreme case, we would then not observe any negative effects of software patenting simply because those developers who have experienced negative ef-

fects are less likely to be included in the sample. Yet another challenge pertains to measuring software patent availability and exposure. While we think that FOSS developers' motivation is primarily affected by (subjectively) perceived patent exposure, irrespective of how strong or weak opportunities for software patenting are in a strictly legal sense, it would be useful nonetheless to develop more "objective" indicators on patent pressure based on systematic legal analysis.

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 on 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 effect or a negative effect on FOSS development.

We have 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 is slower or weaker in areas where patenting in the related proprietary software area is “thicker”; analysis of whether FOSS developers’ participation in terms of code contributions is negatively affected by patents.

We have argued that the third strategy is likely to produce the most interesting results. To that end we have presented a model and empirical research strategy that illuminate 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 and research strategy 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 in the proprietary realm.

6. 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.

Harhoff, D., Wagner, S. (2006). *Modeling the Duration of Patent Examination at the European Patent Office*. University of Mannheim GESY, Working Paper No. 170.

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).

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

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.

Popp, D., Juhl, T., Johnson, D.K. (2004). Time In Purgatory: Examining the Grant Lag for U.S. Patent Applications. *Topics in Economic Analysis & Policy*, (4)1, Article 29.

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.

Samuelson, P., Scotchmer, S. (2002). The Law and Economics of Reverse Engineering. *The Yale Law Journal,* (111), 1575-1663.

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. (2001). Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard Setting. In Jaffe, A.B., Lerner, J., Stern, S. (Eds.), *Innovation Policy and the Economy, Volume 1* (pp. 119-150). Cambridge (MA): NBER.

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.

CHAPTER III

Nothing really matters? Empirical Evidence on the Effects of Software Patents on the Motivation of Free/Open Source Software Developers

I thank Reto Hilty, Luigi Innocente, and Christophe Geiger of the Max Planck Institute for Intellectual Property, Competition and Tax Law (Munich, Germany) for long discussions and helpful comments. I also thank Bob English and Charles Schweik for providing project information for the sampling, and Google, OpenMoko, and ETH Zurich's Neptun program for sponsoring the lottery prizes for the survey.

Abstract

The debate about software patents (SWP) in the EU has been accompanied by massive lobbying by both proponents and opponents. Particularly Free/Open Source software (FOSS) advocates have expressed concerns that SWP will affect FOSS developers negatively. Yet, policy makers have no systematic data that would enable them to decide on this question, which in the past has led to divergence in software patent policies of the US and the EU. This study presents a first attempt to tackle this question empirically by focusing on the effects on developer motivation as one of the key driving forces behind the performance of the FOSS system. Arguments and hypotheses for both sides are formulated and tested against a new data-set from a large, dedicated developer survey. Initial findings are: (1) actual SWP incidents are rare events and the presence of SWP in general has no significant effect on FOSS developer motivation; (2) empirical results do not lend significant support to neither SWP proponents' nor SWP opponents' hypotheses; (3) measuring the presence of SWP in a meaningful way proved to be very difficult; accordingly, three different measures are discussed. The conclusion offers some advice for policy makers and suggestions for further research.

1. Introduction

Many Free/Open Source software (FOSS) advocates have expressed concerns that software patents have a negative influence on FOSS development. This article investigates to what extent this concern is warranted by focusing on the effects of patenting software on FOSS developer motivation. Arguments of proponents and opponents related to intrinsic and extrinsic motivational factors are condensed into five hypotheses. They are then tested using a survey of FOSS project leaders conducted in autumn 2007.

Two trends have been affecting the software industry over recent years. First, Free/Open Source Software (FOSS) has emerged as a dynamic phenomenon: software that is freely available attracts more and more users. Some advocates expect that the unorthodox regime of collaborative development and shared code ownership will largely impact the industry's royalty-based business model and oligopolistic market structure. Second, being able to patent software – "the quintessential digital good" (Quah 2002:29) – presents a case of particular debate for policy makers, researchers, and the proprietary and open source software industry. Opponents consider software patents (SWP) the largest threat to the FOSS community. They fear the opaque SWP system, arguing that it causes legal uncertainty and doubt, which reduces developers' motivation to contribute and may threaten the viability and performance of the FOSS system. Important motivations like joy or self-expression in writing software are adversely affected. Lastly they point out that FOSS projects usually have no income or legal representation, which makes them more vulnerable to lawsuits than proprietary software businesses.

Yet, one may wonder if this fear is really warranted. Why do SWP matter for individuals who reside in different countries and have no formal legal representation, and who furthermore – carefully observant of copyright law – share software code among themselves via servers that can in turn be located in different countries? It does matter, because each contributing developer does reside within the boundaries of a national territory, a jurisdiction that either does or does not allow software to be patented. If it does, the creation, distribution, sale, export, and related activities are bound by patent law, which by definition only gives the patent-holder an exclusive right for such activities. Anecdotal evidence shows that SWP can put restrictions on the distribution of FOSS, but we still lack systematic knowledge of how software patents affect the creation of FOSS by the developer community.

Research has been conducted on the broad motivational spectrum of FOSS developers (see Krishnamurthy (2006) for an overview) and on software patenting in the US (Bessen and Maskin 2006). Only initial theoretical work exists that links both fields (Dapp and Bernauer 2009). This article further investigates the causal mechanisms relating SWP and FOSS developer motivation and presents a first empirical study comparing two regions with significant differences in the presence of SWP.

Two 'worlds' related to SWP are established to serve as a theoretical frame. In the first world SWP have a strong presence: They are easy to obtain and are found in various software domains. The second world features a low presence of SWP: they are more difficult to obtain and the legal backing is weaker. SWP proponents support the first world, as it offers, in their eyes, an effective means to foster innovation by triggering monetary motivation among FOSS developers – much like it does in other domains where patents on inventions can be obtained. In contrast, opponents identify a fundamental incompatibility between how patents function and how software is developed, particularly under the FOSS regime. They argue that non-monetary, joy-related motivation suffers, and thus support the second world with a SWP presence as low as possible.

The US and the EU are good real-world examples of these two SWP worlds: the US features a significantly stronger presence of SWP than the EU. US patent policy and a series of landmark court decisions over the last two decades made it very easy to patent software and the numbers rose significantly (Levine and Saunders 2004; Bessen and Hunt 2004). In contrast, the EU experienced a lengthy debate about 'computer-implemented inventions' over the last several years before the proposal for a related directive by the European Commission was finally rejected in the European Parliament in 2005. Consequently, the SWP policies of both regions are diverging and establishing different legal environments for the FOSS communities to operate. Interestingly, while the 'Free Software Foundation', the 'Open Source Initiative' and many popular FOSS projects that originated in the US are fighting against the country's SWP policy, parts of the EU consider FOSS to be strategic in gaining ground against the dominant US software industry.[19]

Through a survey conducted in autumn 2007 among 1,815 US- and EU-based project leaders from the largest FOSS portal called SourceForge, a new data-set was built. Respondents were asked about their motivations and their experiences with software patents during the time period August 2005-August 2007. The data was used to test the hypotheses put forward in this article.

19 The 'Open Source Observatory and Repository for European public administrations (OSOR)' tracks FOSS adoption across countries, regions, and cities in the EU. See http://osor.eu.

Neither the proponents' nor the opponents' positions are significantly supported by the empirical results. The empirical results do not lend support to proponents' arguments for positive effects on monetary and skills-related motivation. They also do not lend support to opponents' arguments for negative effects on joy- and self-expression-related motivation. Of particular interest was the measurement of SWP presence, for which three approaches are presented and discussed. None, however, proved to be an overwhelmingly good single predictor for FOSS developer motivation.

The introductory section summarizes the software patent debate and describes the FOSS system and what role motivation plays. The following section describes the two theoretical SWP worlds and the arguments of SWP proponents and opponents, leading to hypotheses for each side. The section on research design and methods is followed by the results section, including statistical analysis and interpretation of the results. The final section concludes and offers suggestions for future research.

2. Software patents and Free/Open Source software

2.1 The blurry legal status of software patents

Patents are state-granted, time-bound exclusive rights to prevent others from commercial exploitation of an invention (Lévêque and Ménière 2004). They are built on the assumption that inventors are primarily motivated by money and that they disclose their invention to the public only in return for monetary compensation. When a patent is granted on an invention other implementations of it require permission of the patent-holder, who in return, is obliged to disclose the invention to the public, so that a skilled layperson can replicate it. Furthermore, a patent covers all implementations of an underlying idea. A consequence of this logic is that independent invention can be infringing as well (in contrast to copyright, where only identical copies are infringing). If an independently developed invention falls under the claims of a patent, the developer becomes liable – even if unknowing and unaware.

For three reasons it is difficult to say whether or not software can be patented in a particular jurisdiction. First, there is no commonly accepted definition of what a software patent is. Not only are the semantics of the term disputed but also whether the term itself is adequate. The semantic debate is about what a software patent actually encompasses: is it the software itself ('software patent') or the combination of software and hardware ('computer-implemented invention') as titled by the European Commission (2005)? Second, Cohen and Lemley argue that patent granting practice in the US has created a "doctrine of the magic words" (Cohen and Lemley 2001:9) as patent specifications have been drafted to conceal the fact that they actually refer to software. When drafting patent claims lawyers deliberately use broad language. The applicant's interest to gain broad protection for his invention only reinforces this tendency since a single patent protecting one concept/idea typically covers multiple ways of implementation into software code. Third, there can be a gap between the letter of patent law and the granting practice of patent offices. To give an example: Despite the fact that art. 52 of the European Patent Convention explicitly forbids patenting of software, approximately 70,000 of patents qualifying as software patents have been accorded by the European Patent Office during 1982-2004 (Rentocchini and De Prato 2006), often using similar wording as in the US.

The US is considered the country with the highest presence of SWP. Several policy developments have led to this situation (Jaffe 2000:1) the creation of the US Court of Appeals for the Fed-

eral Court pools patent cases at one place; 2) the Baye-Dohle Act allows patentability of federally funded R&D; 3) the WTO agreement on trade-related aspects of intellectual property rights permits patentability of "all fields of technology" (TRIPS, art. 27); 4) key court decisions have gradually expanded patentable subject matter to include software and business methods (Levine and Saunders 2004). As a result, the number of US software patents has increased significantly (Bessen and Hunt 2004): until 2001 roughly 80,000 US software patents had been granted. Cohen and Lemley (2001:3) state that the question of whether or not software is patentable in the US is a matter "for the history books" and no debate anymore. In contrast, the European Parliament rejected a proposed directive on 'computer-implemented inventions' in 2005, after long debate and massive lobbying on both sides (Haunss and Kohlmorgen n.d.). Despite that difference, SWP have been granted by the EPO; but their legal enforceability is much less clear than in the US. In sum, SWP presence is stronger in the US than in the EU: while it is intentional in the US, it is rather a faux pas in the EU, as the legal frame there would suggest a different situation.

2.2 The FOSS system and the motivation behind

The way software is produced, managed, and distributed under the FOSS system is very different from the rules and processes known from the proprietary part of the industry. Three distinct features of FOSS cause this difference (Dapp and Bernauer 2009). Firstly, enabled by the Internet and affordable personal computers, large numbers of developers engage in *massive decentralized, collaborative development of software*. Developers regularly exchange source code from FOSS projects. Most projects of a certain size adhere to a meritocratic governance structure that can be quite sophisticated but is not comparable to typical firm hierarchies since developers freely choose if and how they contribute. No formal contract relations exist, and often the project does not even have a legal status. Geographically, developers and servers hosting source code can be located in different jurisdictions. Secondly, developers license their source code under an *open source copyright license*. Everybody is allowed to run, copy, modify, and redistribute the software as they see fit. Thus, developers give up most of the control over the code that traditional copyright grants them. Having many developers contribute under this regime leads to a system of shared copyright and ownership that prevents effective control by one individual. Some licenses include a so-called 'copyleft' mechanism to prevent users from appropriating the code in proprietary software. Thirdly, open source copyright licenses require *source code to be freely available* for everybody, usually downloadable from the Internet. As a consequence, there is no royalty-based revenue stream from license sales for developers.

If there is no money to be made, why do developers contribute to FOSS at all? They do it because their motivation is broader than money alone. Ryan and Deci (2000) call motivation intrinsic when something is done because it is inherently interesting or enjoyable, for instance, because of the fun or the challenge it involves. Extrinsic motivation is present when someone acts because such activity leads to an additional, separate outcome – an instant reward or an incentive with a time delay.

The role of motivation was one of the first topics FOSS research tackled: Krishnamurthy (2006) provides a synopsis of several empirical studies investigating the motivational setup of FOSS developers. He concludes that both intrinsic and extrinsic components of motivation are important and finds that "the evidence is mixed on the relative value of intrinsic and extrinsic motivational components" (ibid., 27). Different studies support this heterogeneous picture (Ghosh *et al.* 2002; Hars and Ou 2002; Hertel *et al.* 2003; Lakhani and Wolf 2005; Roberts *et al.* 2006).

TABLE 1 provides an overview of all motivational factors that Dapp and Bernauer (2009, Table 4) identified as potentially affected by SWP. To meet space constraints in this article, a representative selection of five factors, marked (*), was made for further analysis: MONEY and SKILLS in the group of extrinsic factors relate to the main aspects of SWP, monetary incentive and knowledge disclosure. From the group of enjoyment-based intrinsic factors, JOY as the most important motivational factor overall (FIGURE-A 1) and SELF-EXPRESSION as the most directly affected factor by SWP restrictions have been selected. From the group of obligation-based factors, SOFTWARE FREEDOM was selected because one can argue for effects of SWP in both directions and developers motivated by 'software freedom' may be more sensitive towards SWP issues than other FOSS developers.

Software patents are feared by FOSS developers since they may limit their freedom to write software – up to a point where they can block development completely. The Free Software Foundation (2008) and the Foundation for a Free Information Infrastructure (2009) maintain a list of cases where SWP had negative effects on open source or proprietary software programs.[20] The risk is considered high because many patents have already been granted and a single broad enough patent could block broad areas of functionality, resulting in many infringing programs at once. As a consequence, developers would have to pay license fees or make do with less functionality.

20 See http://www.gnu.org/patent-examp/patent-examples.html and http://eupat.ffii.org/patents/effects/, retrieved 13.03.2009.

TABLE 1: motivational factors potentially affected by SWP

Instant Rewards (extrinsic)	Enjoyment-based (intrinsic)
SKILLS (*). Learn new coding skills by reading and writing code.	JOY (*). Programming as a fun activity, such as a hobby.
SELF-HELP. Ability to help oneself by improving programs.	SELF-EXPRESSION (*). Ability to express oneself aesthetically through software code. Code writing as an art form.
NET GAIN. A developers gets access to the whole program, although he contributes only small parts.	ALTRUISM. Provide freely available useful software to support other users.
MONEY (*)(M). Direct monetary reward, e.g., when being hired to write code for a FOSS project.	CHALLENGE. The intellectual challenge of solving difficult programming problems.
Delayed Incentives (extrinsic)	**Obligation-based (intrinsic)**
CAREER (M). Signal skills to potential future employers, self-marketing.	IDENTIFICATION. Identification and belonging to a community.
	NORMS. Observance of community norms like, e.g., sharing.
	SOFTWARE FREEDOM (*). 'Software must be free' as a political mission.

Based on Dapp and Bernauer 2009.

Anecdotal evidence aside, however, we have very little systematic knowledge about the effects of software patents on FOSS developers' motivation. We lack theoretical work investigating how the presence of SWP influences certain motivational factors and we lack empirical studies testing such hypotheses. Does motivation differ for developers who face different levels of SWP presence? If, controlling for other factors, correlations can be observed empirically, they will merit further research since motivation is an important cornerstone of the FOSS system.

3. Software patent presence and FOSS motivation

For the 'model worlds' with differing levels of SWP presence, I define SWP presence on two levels: availability and prevalence. While SWP availability relates to the legal opportunity to patent software, SWP prevalence relates to the willingness of the players to make use of that opportunity, leading to varying patent presence and SWP incidents. Proponents of SWP should be interested in increasing the presence of SWP, while opponents are interested in decreasing it. This section presents the arguments on both sides.

3.1 The proponents view

"Patents play useful roles in the open source environment." (EICTA 2000:4)

Having strong confidence in patents as effective monetary incentive instruments, proponents support a high-SWP-presence world. They argue that the FOSS field should not be excluded, as it is not different from other industries in terms of patentability: Patents work in FOSS as well as in other innovation fields. Therefore, they are in favor of extending the patent system to FOSS and they expect a positive effect on extrinsic, especially monetary, motivation of FOSS developers.

Two of the three classical utilitarian/economic patent theories are related to money (Fisher 2005). *Incentive theory*, "probably the most quoted argument in favor of patents" (Dutton 1984:20), argues that patents enhance innovation because they give an incentive to the individual to make the needed up-front investments to invent and later commercialize the invention (Mazzoleni and Nelson 2004). *Reward theory* argues that patents "secure inventors their just reward, proportional to the usefulness of the invention to society" (Fisher 2005:8). Patents increase the profit of the inventor, discourage competitors from free-riding, and therefore help to prevent imitation. Inventors deserve an exclusive right since an invention's potential to perform tasks more efficiently or satisfy needs more effectively represents a measurable cost-saver to users.

In the FOSS system, SWP proponents argue that developers also make considerable investments when writing software, but in contrast to proprietary software producers, they have no license income to compensate for their time commitment. Here, SWP present a means of compensation and an incentive for them to continue. Consequently, FOSS developers should be more motivated by money when they reside in regions with a high presence of SWP.

HYPOTHESIS 1: A high presence of SWP increases developers' monetary motivation to contribute to FOSS projects.

Exchange theory, the third classical utilitarian patent theory, argues that patents offer a fair balance between the interests of the public and the inventor (Fisher 2005): Through the patent letter the public gets access to the knowledge embedded in the invention, while the inventor gets exclusive rights in exchange for disclosing the invention. Through the construction of a time-bound exclusive monopoly right that balances the interests of the inventor and the public, patents encourage both the invention and its publication.

In the FOSS system, proponents (EICTA 2000:4) argue that a SWP discloses underlying concepts of a software invention in a way source code cannot. While source code is low-level and detailed, patent letters offer conceptual knowledge in a form that is more appropriate for learning about the invention, identifying directions for future code development and thus learning new skills. Therefore, the level of skills-related motivation should be higher among FOSS developers residing in places with a high presence of SWP.

HYPOTHESIS 2: A high presence of SWP increases developers' skills-related motivation to contribute to FOSS projects.

3.2 The opponents view

> *"Software patents are the software project equivalent of land mines: each design decision carries a risk of stepping on a patent, which can destroy your project." (Stallman 2004)*

Being skeptical about whether the patent logic is compatible with the FOSS development process, opponents support a low-SWP-presence world. They point out that the FOSS field should be excluded, as it is fundamentally different from other industries in terms of patentability: in their view, SWP fail not only in FOSS but also in the proprietary software industry. Empirical studies showing that 26% of US patent lawsuits in 2002 involved software patents indicate that there is a problem (Bessen and Meurer 2008, ch. 9). They argue that the logic of patents – to protect a single idea as broad as possible – is incompatible with software, because software contains thousands or millions of ideas. Since software is built incrementally by re-using many existing ideas, patent infringements would be the rule rather than the exception. Therefore, opponents are in favor of limiting the reach of the patent system, while they expect a negative effect of SWP on intrinsic, especially joy-based, motivation among FOSS developers

Following the incompatibility argument, SWP have the potential to slow down innovation or even block software development in the FOSS, as well as in the proprietary software industry. In the FOSS field, however, the threat is higher and the defense weaker. FOSS projects are more vulnerable for several reasons: first, open access to source code makes infringements easier to detect. Second, FOSS projects have very little legal backing. SWP are perceived as a risk that is hard to identify and quantify because detecting infringement is difficult. In addition, many developers are volunteers for programming: researching patent databases for potential infringements in order to assess the legal risk exposure is something hardly any FOSS developer would do. As a consequence, the opaque SWP system causes fear. Third, individual developers have no effective defense strategy. They could set up organizations dealing with legal issues, but building up bargaining power by creating own patent portfolios is practically impossible for lack of knowledge and financial resources. For proprietary companies this is not an issue: Bill Gates of Microsoft wrote in 1991 that "[i]f people had understood how patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be at a complete standstill today." He suggested, that "[t]he solution to this is patent exchanges with large companies and patenting as much as we can." (Memo "Challenges and Strategy", 16. May 1991).

In sum, writing software in a patent-dense domain exposes the developer to incalculable legal risk, with few or no defense options. The fear of becoming legally liable affects the joy developers experience from contributing and sharing code. Thus, we would expect the level of joy-related motivation to be lower when SWP presence is high. This may be so where SWP can be obtained easily and/or developers operate in a patent-dense software domain where infringement claims are often raised.

HYPOTHESIS 3: A high presence of SWP decreases developers' joy-related motivation to contribute to FOSS projects.

A further implication of the potential of SWP to block software development is the restriction imposed on developers to freely formulate code details. Dapp and Bernauer (2009) note that one important motivation for FOSS developers is to write not only functional, but also elegant or aesthetic code. A SWP traps many possible code implementations by broadly protecting a single idea, which restricts developers' artistic freedom of self-expression and to formulate code as they deem best.

HYPOTHESIS 4: A high presence of SWP decreases developers' self-expression-related motivation to contribute to FOSS projects.

3.3 Do software patents really matter for FOSS?

"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."
President, UK Institute of Civil Engineering 1851 (Fisher 2005:15)

Despite the arguments discussed above, one may think of reasons why the presence of SWP may be inconsequential for developer motivation. First, there may be no "need to stimulate invention" (Fisher 2005:14) in a monetary sense. Positive effects of SWP on monetary motivation may not be present since a substantial amount of motivational factors driving the FOSS system are non-monetary, such as "joy" (Krishnamurthy 2006). Second, there may be no observable negative effect on joy or self-expression because developers do not experience an actual threat. Patent holders may not hunt down infringing developers because there are too many and they are not a financially worthwhile target. As long as infringement remains unsanctioned, patents may have no observable negative effect. Third, how a FOSS developer deals with a software domain cluttered with patents is not clear per se. Depending on the self-conception of the developer, he may belong to one of two camps within the community: politically motivated 'software freedom fighters' or 'open source pragmatists'. The first group is more motivated by software freedom and thus develops software to provide free alternatives. The second group is less motivated by political motives but rather by pragmatic aspects. Such a developer may be more inclined to withdraw his engagement and avoid certain software domains when facing patent issues. Therefore, one may expect a 'freedom fighter' to target his efforts at patent-cluttered areas – precisely to provide patent-free alternatives here as well. These developers search the "dangerous neighborhood" of patented technology to provide an escape: an equivalent patent-free solution that is available to everybody as FOSS. Ogg Vorbis, a patent-free open-source audio file format, is an example of a technology invented to avoid a patent-covered technology called MP3. In other words, a high SWP presence potentially attracts developers with a high software-freedom-related motivation.

HYPOTHESIS 5: A high presence of SWP attracts developers with a high software-freedom-related motivation to contribute to FOSS projects.

Which camp dominates is an empirical question, and one that is hard to answer since projects that effectively ceased because software patent issues made developers leave are very hard to identify. Potential consequences of this selection effect are discussed in the next section.

4. Research design and methods

4.1 Data collection and sampling strategy

The online survey targeted leaders of FOSS projects registered on SourceForge.net (SF) to find out about their experiences with SWP. FOSS developers who held a project leader function were selected for several reasons: First, SWP can only be assigned to natural or legal persons, and many projects are not represented by a legal person. If a project is attacked, the project leadership will be first to know. Second, having been a developer who has contributed code in the past, he or she is well equipped to answer questions related to individual motivation. Third, having the role of a project leader, he or she should also be able to answer questions related to the project as a whole.

SF was chosen because it is the largest FOSS developer community portal. It offers a broad variance in geographical location of developers and in software application domains that may also be relevant to the SWP question. In addition, employment status also varies for SF developers. Large 'top shot' projects (e.g., Linux kernel, Apache web server, OpenOffice.org productivity suite), with their own websites, have not been included in the sample. The developer community of such projects is less representative, as many work for companies sponsoring these projects as part of their business model. Software domain variance would be limited, as one large project usually covers only very few software domains, often only one. These projects also tend to be 'over-surveyed', as many studies have focused on them in the past.

Two sources have been useful in building the sample. First, the main problem with SF – abandoned or 'dead' projects – had to be countered by setting up a well-defined sampling process that excluded such projects. English and Schweik (2007), who developed a metric to separate failed and successful projects using release numbers and time intervals between releases, kindly provided me with a list of 57,085 names of 'alive' projects that had produced at least one code release by August 2006. Second, I chose FLOSSmole (Howison *et al.* 2006), a data collection project led by Megan Conklin at Syracuse University, which provides a central repository uniting data from different FOSS portals, including SF to extract information about project leaders.[21] Thus, the sample frame included project leaders of all 57,085 'alive' projects hosted on SF as of August 2006. From these a random sample of 11,000 was drawn and invited to a web-based survey. In total,

21 See http://ossmole.sourceforge.net/.

2,441 individuals responded (22% overall response rate), of which 1,815 resided in the US or the EU.

'Failed' projects were not considered for the survey because abandoned projects are much less likely to respond and provide reliable answers. Hence, the sample does also not include developers who abandoned projects or who never got involved in a project because of SWP issues. An ideal sample would of course include past, current, and future FOSS developers. That being impossible for practical reasons, the question is in which ways does this selection effect distort the sample? Having only 'motivated' developers who were active during the survey period means that developers with low motivation levels are underrepresented in the sample. As a consequence, statistical results based on this sample will give more support to hypotheses arguing for increasing motivation levels (H1, H2) and give less support to hypotheses arguing for decreasing motivation levels (H3, H4) than they actually should. In other words, statistical analysis will report smaller effects than there actually are. (The effects on H5 are more complicated and will be discussed in the results section.) If SWP incidents are rare events as argued earlier, there should not be much sample distortion, as only a very small number of developers would have abandoned their projects due to SWP. The finding by Arundel et al. (reported in Ghosh and Aigrain (2006:121)) – that 10% of firms in ICT producing and ICT- intensive sectors change or avoid lines of research because of concerns that others have patents in that area – may serve as an upper boundary. In any case, the decision to fully abandon a project is the action of last resort for a FOSS developer. Up to that point, SWP have the potential to hinder or delay the development process and accordingly affect the motivation of the involved developers.

4.2 Survey and questionnaire

A questionnaire was designed and tested in two pilot phases preceding the main survey run. A lab pilot among 20 researchers and FOSS developers was used to get feedback on the online setup as well as on questions of wording and answer categories (Fowler 2002; Fowler 1995; Dillman 2000; Czaja and Blair 1996). Subsequently, a field pilot with a sub sample of 1% of the target sample was run with a first questionnaire: this pilot collected answers on a few open questions that were clustered and recoded into multiple choice questions for the main run.[22] In autumn 2007, the main survey was run in three parallel batches of one-week intervals to reduce the risk of a technical failure stopping the survey process. Over four weeks, respondents of every batch received four

22 Only answers from the field pilot (response rate 24.5%) were included in the final data-set. Data from the lab pilot was not used.

emails: one invitation, two reminders, and a last call. As a result, the main survey took six weeks to run.

A general challenge in surveys is to avoid sample distortion by biased respondents. The main cause of such distortion is self-selection of participants. To avoid the practice used by previous developer surveys of recruiting participants through a 'snow-ball'-like system, where the set of potential respondents is not defined ex ante, a controlled survey environment was set up, with a pre-defined sample frame (see above), simple random sampling, targeted personal invitations, and mechanisms to prevent non-invitees from participating. These measures helped minimize coverage and sampling errors (Dillman 2000:9).[23]

4.3 Key variables

The *dependent variables* are motivational factors. Respondents had to rank the importance of each motivational factor between 'not important at all' to 'very important' on a 6-point Likert scale. The following factors (see TABLE 1) were analyzed: MONEY, SKILLS, JOY, SELF-EXPRESSION, and SOFTWARE FREEDOM. FIGURE-A 1 in the appendix shows the means of all motivational factors.

On the predictor side, SWP presence is the key explanatory factor.[24] The concept is represented by three different variables. (a) Accounting for the intricacies of patent laws across countries would require dedicated legal studies for each country to construct a comparative index measuring the strength of the SWP regime. A simplified classification was used instead: a high presence of SWP in a jurisdiction was coded as '1', the low presence as '0'. Therefore, the US (and Japan) was assigned a '1' for the highest SWP presence, the EU and remaining countries a '0'. Levine and Saunders (2004) and Bessen and Hunt (2007) provide detailed support for these assignments. (b) SWP prevalence is measured in two ways: first, SWP incidents can only occur if the jurisdiction provides legal ground for them. Thus, respondents have been asked whether the project they led had faced any SWP incident. The according yes/no dummy was included as INCIDENT. (c) The second measure for SWP prevalence is bound to the software domain of the FOSS project. Anecdotal evidence suggests that certain software domains, like cryptography or multimedia,

23 Instead of using a mailing list or website to announce the survey, a project leader for each project was identified (using the FLOSSmole database) and invited by email. Invitations contained a personalized web link to prevent non-invitees from participation. After the respondent had completed the questionnaire, the questionnaire was closed and further access denied. (LimeSurvey was used to manage the survey.) A lottery with gadgets attractive for the target group was offered to counter possible selection bias that only 'SWP-haters' would participate.

24 Based on the concept that 'software patents' are inventions embodied in software (Allison and Lemley 2000), I use the definition by Bessen and Hunt: software patents cover 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).

face higher SWP prevalence than other domains. From the list of 14 domains by which projects hosted on SF are categorized, respondents were asked about the top four domains with highest patent pressure. From this an index for patent pressure was constructed (TABLE 2) and grouped into three pressure levels: High, low, and no domain pressure. The first two are included as dummy variables: HI DOM PRESS and LOW DOM PRESS.

The following *control variables* are included: (a) SWP RESEARCH indicates whether or not a project conducted research into the patent situation in their field; (b) whether a developer has direct (by coding) or indirect (other FOSS-related work) income from FOSS is captured in the dichotomous variable FOSS INCOME; (c) HIGHER EDU captures whether or not a respondent has an advanced education degree (bachelor or higher); (d) experience is measured in two dimensions: in terms of years of engagement (EXP [yrs]) and number of projects contributed (EXP [pjs]); (e) information on age is represented in a simplified way by cutting the age group curve (FIGURE-A 2 in the appendix) at the median category: AGE>30 flags when respondent was above 30 years old. Gender differences were insignificant during analyses and have been omitted.

5. Empirical results

5.1 Descriptive statistics

This section presents descriptive statistics about the dependent variables (five motivational factors), the predictor variables used to measure SWP presence (law, patent pressure by domain, and SWP incidents) and some background information about the survey population (employment and income situation).

The survey covers the 2-year period from August 2005 to August 2007. During that time, 637 (26,1%) respondents resided in the US, 1,178 (48,3%) in the EU, and 626 (25,7%) in the 'rest of the world' (ROW), yielding a total of 2,441 respondents (response rate 22%). The two dominant age groups (26-30 and 31-35 year-olds) mark the top of the bell curve (c.f. FIGURE-A 3 in the appendix) and make up for 51.5% of the sample – 16.4% are younger and 32.1% are older. The majority of respondents are male (1.5% females) and well educated (81.8% with a university degree), which is in line with the 'FLOSS study' that reported 70% developers with a university degree and a ratio of 1.1% females (Ghosh *et al.* 2002).

Figure 1 shows the aggregated importance levels (mean values) of the five motivational factors, broken down by region. *Across* all regions, joy-related motivation has the highest, monetary motivation the lowest overall level. Motivation related to self-expression and skills share the second position and software freedom is in third place. This suggests the relative importance of intrinsic over extrinsic factors compared to previous studies in which Hars and Ou (2002) stressed extrinsic factors; Lakhani and Wolf (2005) found intrinsic factors to be important, whereas Roberts *et al.* (2006) found no impact of intrinsic motivation on code contribution. We still miss a clear picture of the motivational FOSS landscape.

Almost identical levels *within* regions (except for money earning and software freedom) would suggest that the motivation levels do not differ much between them. Yet, further investigation is appropriate since these aggregated values do not show all possible influencing factors. The geographical breakdown only captures the legal availability of SWP, not all differences in SWP presence. Focusing only on the geographical location may not be enough; the software domain a program belongs to may be a strong predictor as patent pressure may vary more across software domains than across jurisdictions.

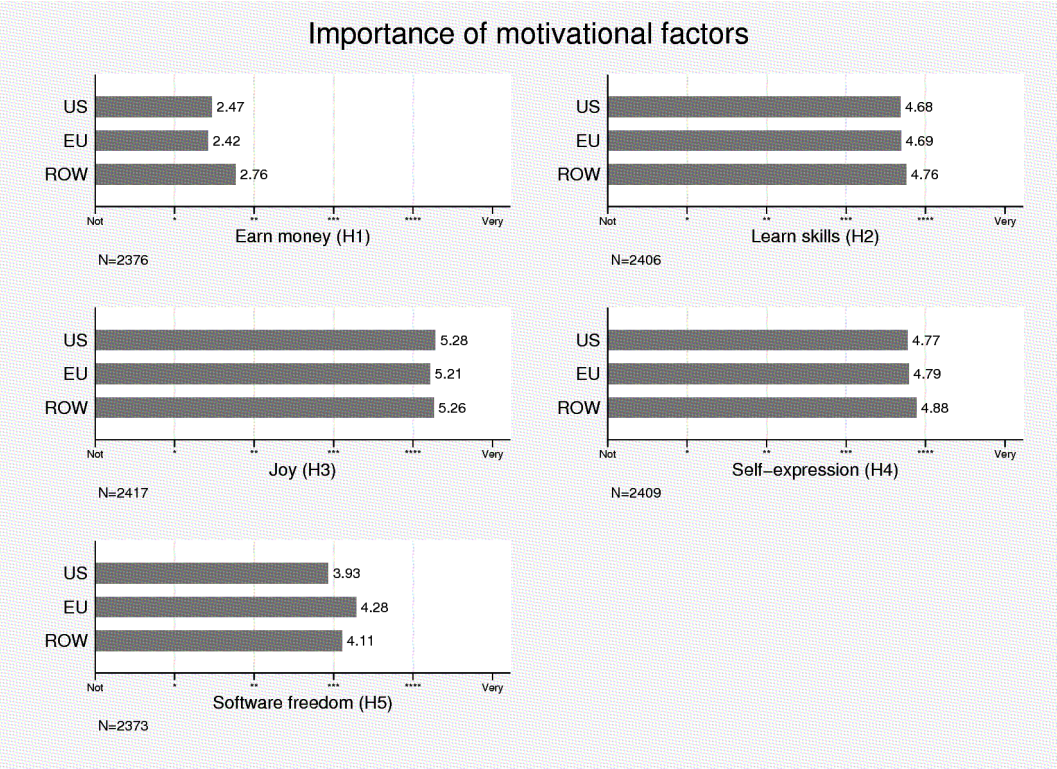

FIGURE 1: *Relative importance of motivational factors by region (mean values)*

TABLE 2 shows the index constructed to measure patent pressure across software domains. The index values are shown together with the share of sampled projects. One can see that respondents perceive a moderate patent pressure in most software domains, except the last three.

Two cut points have been identified to divide the software domains into high pressure (index >0.5), low pressure category (0.1<index<0.4) and no pressure (index <0.1) categories. These categories are used in the statistical analysis.

TABLE 2: *Patent pressure index by software domain*

	Software domain	Pressure	Share (%)
[HIGH]	Multimedia	0.513	170 (7.0)
	Formats/Protocols	0.505	32 (1.3)
[LOW]	Office/Business	0.342	98 (4.0)
	Communications	0.318	148 (6.0)
	Security	0.296	64 (2.6)
	Software development	0.261	390 (16.0)
	Science/Engineering	0.260	257 (10.5)

	Internet	0.257	351 (14.4)
	System	0.256	160 (6.6)
	Desktop	0.220	62 (2.5)
	Database	0.172	96 (3.9)
	Game/Entertainment	0.170	282 (11.6)
[NO]	Education	0.044	79 (3.2)
	Text editors	0.019	29 (1.2)
	Other	0.000	223 (9.1)
	(total sample)		2,441 (100.0)

The third predictor variable for SWP presence measures SWP incidents. To better identify differences between perception of SWP and actual incidents, respondents were also asked about how they perceived the *role* SWP played for their project (positive, none, negative). TABLE 3 shows a cross tabulation of the SWP incident dummy with this auxiliary perception variable 'SWP role', broken down by region.

TABLE 3: *Actual patent incidents vs. perceived role of SWP, by region*

Patent role	**US**		**EU**		**ROW**		**Total (%)**
SWP incident?	*No*	*Yes*	*No*	*Yes*	*No*	*Yes*	
Positive	6	0	12	2	9	0	29 (1.7)
None	362	1	683	5	349	5	1,405 (85.0)
Negative	54	11	86	14	49	6	220 (13.3)
Total	422	12	781	21	407	11	1,654 (100)

N<2,441 because respondents had the option of not answering these questions.

An incident – a holder of a SWP approaching the project – was reported in 44 of 1,654 cases, making SWP incidents a rare event (2.7%). That remarkable result is consistent across all regions (EU 2.8%, US 2.7%, ROW 2.7%) meaning that the probability for a SWP incident is largely independent of the jurisdiction and its patent law. A low SWP presence is no safeguard for fewer incidents, while a high presence does not necessarily lead to more incidents.

Despite an equal incident risk-level on this aggregation level, seven times more respondents perceived SWP to have played a negative (13.3%) rather than a positive (1.7%) role for their project, while the large majority (85%) stated SWP did not play any role at all. (Two EU respondents reported a positive role, although they had one incident. The explanation is that both

projects had patents of their own and belonged to a protection scheme/patent pool providing legal defense.)

An important aspect when analyzing motivation and patents is the employment and income situation. How were the respondents employed, and did they earn money related to FOSS? Figure 2 shows employment situation and income source. The majority (86.1%) was employed or self-employed and only 11.5% were unpaid students (N_{tot}=2,429). For those earning money, the income source is mostly *not* connected to FOSS (69-76%, depending on region). This is a higher value than Ghosh *et al.* (2002) reported, where 46% of developers did not earn money from FOSS. Conversely, only 9-11% of the employees earned money from FOSS-related work (coding and other). This is considerably lower than the 40% "paid contributors" that Lakhani and Wolf (2005) found in their study and the 50% of directly paid, salaried or contract developers reported by Hars and Ou (2002).

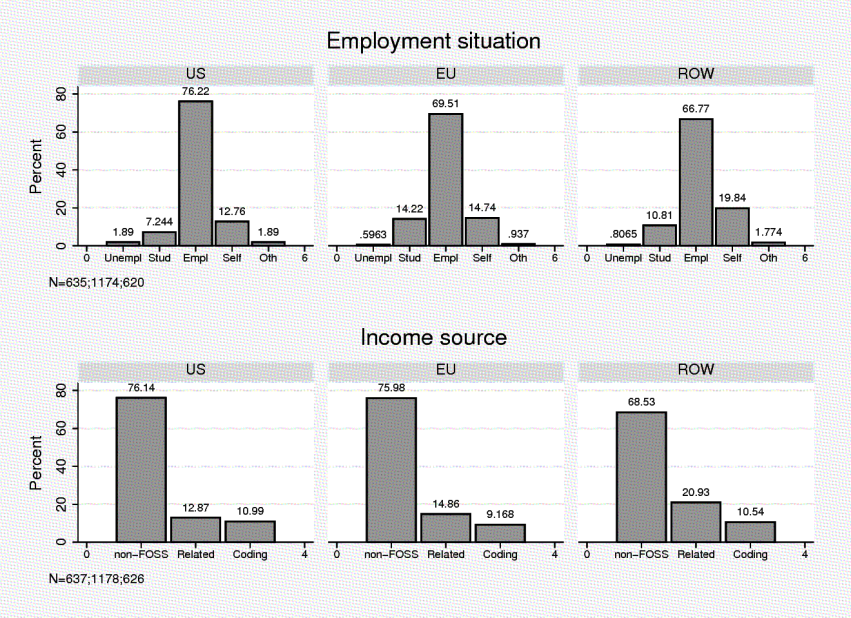

FIGURE 2: Employment and income situation by region

5.2 Regression analysis

Logistic regression was used to model the specific effects of SWP presence on motivation. Missing values in the dependent variables and predictors were imputed using multiple imputation procedures for Stata (Allison 2002; Royston 2005). See part E of the main appendix for complete imputation statistics.

The dependent variables are the five motivation variables MONEY, SKILLS, JOY, SELF-EXPRESSION, and FREEING SOFTWARE. In the survey, they were ordinal-scaled on a 6-point Likert scale ranging from 'not important at all' up to 'very important'. For the analysis, they are dichotomized at their means, yielding two categories. Placing the cut-off point between category 3 and 4 would assume that respondents have similar interpretations of what the six categories denote. Cutting at the mean does not make this assumption, but instead indicates above- or below-average motivation for each factor using 1 or 0.

The concept *SWP presence* is represented by three different predictors yielding three models for each hypothesis test. First, *SWP LAW* proxies the legal situation regarding software patents in the US and the EU, equaling '1' if SWP are legally available (US), '0' if otherwise (EU). (For other jurisdictions this variable could not be defined. Those observations have been omitted for this analysis, reducing the number to N=1,815.) Second, *HIGH DOM PRESS* and *LOW DOM PRESS* are the top two levels of an ordinal variable indicating the patent pressure on the software domain the respondent's project operates in based on the index described in TABLE 2 (the reference category 'no pressure' is omitted from the regression). Third, *INCIDENT* is a dummy for patent incidents, equaling '1' if the project had a SWP incident during the survey period and '0' if otherwise.

The following control variables are not part of the theoretical argument, but may influence the motivations considered: SWP RESEARCH flags whether the project conducted research into SWP in patent databases or not. FOSS INCOME signals when the income source is related to FOSS; equals '1' if respondents had income from coding FOSS or non-coding work related to FOSS as shown in Figure 2. FOSS experience is measured in terms of years, EXP [yrs], and number of projects, EXP [pjs] (FIGURE-A 2). HIGHER EDU flags advanced education; equals '1' if respondents had a bachelor, master, or Ph.D. (FIGURE-A 3). AGE>30 flags when respondents are older than 30 years, the median category (FIGURE-A 3).

TABLE 4 reports parameter estimates as odds ratios for the proponents' hypotheses that a high SWP presence increases developers' monetary (H1) and skills-related (H2) motivation to contribute to FOSS projects.

TABLE 4: Proponents: logistic regression for MONEY and SKILLS

	MONEY (H1)			*SKILLS (H2)*		
	A	*B*	*C*	*A*	*B*	*C*
SWP LAW	1.038 (0.121)			1.056 (0.119)		
HI DOM PRESS		1.463 (0.358)			1.228 (0.278)	
LOW DOM PRESS		1.212 (0.199)			1.466 (0.219)**	
INCIDENT			0.683 (0.286)			1.232 (0.574)
SWP RESEARCH	1.410 (0.251)*	1.393 (0.250)*	1.440 (0.257)**	1.226 (0.213)	1.260 (0.219)	1.217 (0.214)
FOSS INCOME	5.135 (0.657)***	5.171 (0.665)***	5.164 (0.663)***	0.886 (0.108)	0.878 (0.107)	0.885 (0.108)
HIGHER EDU	1.135 (0.167)	1.125 (0.166)	1.133 (0.166)	0.712 (0.104)**	0.713 (0.105)**	0.712 (0.104)**
EXP [yrs]	0.991 (0.016)	0.992 (0.016)	0.992 (0.016)	0.962 (0.014)**	0.963 (0.014)**	0.963 (0.014)**
EXP [pjs]	1.183 (0.098)**	1.176 (0.097)**	1.183 (0.098)**	1.134 (0.093)	1.131 (0.092)	1.129 (0.092)
AGE >30	1.089 (0.132)	1.103 (0.133)	1.095 (0.132)	0.703 (0.081)***	0.708 (0.081)***	0.706 (0.081)***
AIC	2041	2040	2040	2123	2119	2123
BIC	2085	2090	2084	2167	2168	2167

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (e^{β}) with robust standard errors. N=1815.

H1: None of the predictor variables for SWP presence show a significant influence on developers' motivation to earn money. When gaining income from FOSS, the odds to be motivated above average by money are expected to increase by a factor of 5, holding all other variables constant. More project experience increases the odds by 18%, and doing SWP research increases the odds by 41%. In sum, the results do not lend support for H1: factors related to SWP presence have no observable effect.

H2: Only coding in an environment with low patent pressure (compared to no pressure at all) has a significant effect on respondents to learn new skills: it increases the odds by 47%. By contrast, the odds to be motivated above average are reduced by higher education (factor 0.71), more project experience (0.96) and growing age (0.71). The result gives light support for H2, based on patent pressure. SWP law or incidents are not significant.

TABLE 5 reports estimates for the opponents' hypotheses that a high SWP presence decreases developers' joy-related (H3) and self-expression-related (H4) motivation to contribute to FOSS.

TABLE 5: *Opponents: logistic regression for JOY and SELF-EXPRESSION*

	JOY (H3)			*SELF-EXPRESSION (H4)*		
	A	*B*	*C*	*A*	*B*	*C*
SWP LAW	1.268 (0.139)**			1.072 (0.120)		
HI DOM PRESS		0.905 (0.204)			1.383 (0.319)	
LOW DOM PRESS		1.043 (0.157)			1.384 (0.208)**	
INCIDENT			1.036 (0.448)			1.293 (0.618)
SWP RESEARCH	0.704 (0.117)**	0.724 (0.121)*	0.712 (0.119)**	1.179 (0.210)	1.190 (0.214)	1.169 (0.209)
FOSS INCOME	0.670 (0.079)***	0.668 (0.079)***	0.671 (0.080)***	0.951 (0.117)	0.948 (0.117)	0.948 (0.117)
HIGHER EDU	0.832 (0.113)	0.832 (0.113)	0.829 (0.112)	0.993 (0.139)	0.990 (0.139)	0.991 (0.139)
EXP [yrs]	1.020 (0.016)	1.025 (0.015)	1.025 (0.015)	1.007 (0.016)	1.009 (0.016)	1.009 (0.016)
EXP [pjs]	1.358 (0.114)***	1.337 (0.110)***	1.335 (0.110)***	1.096 (0.090)	1.090 (0.089)	1.090 (0.089)
AGE >30	0.675 (0.075)***	0.688 (0.076)***	0.690 (0.076)***	0.718 (0.083)***	0.727 (0.084)***	0.722 (0.083)***
AIC	2212	2218	2217	2102	2099	2102
BIC	2256	2268	2261	2146	2149	2146

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (e^{β}) with robust standard errors. N=1815.

H3: Legal availability of SWP increases the odds for joy-related motivation to be above average by 27%. In contrast, SWP research, FOSS income, and age over 30 years all reduce the odds by a factor of 0.71, 0.67, and 0.68 respectively. In contrast, project experience increases the odds for 'joy' to be above average by 35%. The result shows an opposite relation as predicted by H3. This counterintuitive result may stem from the over-representation of high 'joy' levels due to the selection bias in the sampling. In addition, the variable SWP LAW may pick up more than just SWP differences between the US and the EU, including a cultural bias that makes US respondents report higher motivation levels than EU respondents.

H4: A low patent-pressure environment increases the odds for self-expression-related motivation to be above average by 38%. Being over 30 years old decreases the odds by a factor of 0.72. The results do not lend support to H4.

TABLE 6 reports the parameter estimates for the hypothesis that a high SWP presence attracts developers with a high software-freedom-related motivation to contribute to FOSS projects.

TABLE 6: *Logistic regression for SOFTWARE FREEDOM*

	SOFTWARE FREEDOM (H5)		
	A	*B*	*C*
SWP LAW	0.670 (0.074)***		
HI DOM PRESS		0.776 (0.174)	
LOW DOM PRESS		0.918 (0.139)	
INCIDENT			1.369 (0.592)
SWP RESEARCH	1.116 (0.180)	1.108 (0.181)	1.074 (0.176)
FOSS INCOME	1.393 (0.165)***	1.380 (0.162)***	1.381 (0.162)***
HIGHER EDU	0.660 (0.089)***	0.672 (0.090)***	0.669 (0.090)***
EXP [yrs]	1.002 (0.015)	0.995 (0.015)	0.995 (0.015)
EXP [pjs]	1.183 (0.092)**	1.220 (0.094)**	1.214 (0.094)**
AGE >30	0.773 (0.086)**	0.743 (0.082)***	0.747 (0.082)***
AIC	2220	2234	2233
BIC	2264	2284	2277

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (eβ) with robust standard errors. N=1815.

H5: Legal availability of SWP decreases the odds for above-average software-freedom-related motivation by a factor of 0.67. The odds for above-average motivation are also reduced by having an advanced degree (0.67) and being over 30 years of age (0.75). FOSS income and experience in projects, however, increases the odds to be motivated above average by 39% and 20% respectively.

Further analysis of the sample reveals that 36.6% of all respondents identify with the pragmatist 'Open Source' and 20.8% with the ideologist 'Free Software' camp (42.6% make no distinction/belong to both), but 49.4% of the Free Software camp reside in the EU and only 16.8% in the US. If we assume that developers did not switch countries because of the SWP situation, it would explain that residing in the US – which equals having SWP LAW=1 – leads to a reduced motivation level. The variable 'SWP Law' may also include other, e.g., cultural, differences between the US and the EU unrelated to the legal SWP situation that led to the turn in direction.

6. Conclusion

This study offers first insights into the largely unexplored question of how software patents affect the motivation of FOSS developers. The results can be summarized as follows: First, software patent incidents seem to be very rare events: only 2.7% of the respondents reported such an incident. Second, software patents do not appear to show a strong effect on FOSS developer motivation in general. The presence of software patents has no positive effect on monetary and skills-related motivation, as argued by proponents. Third, it also does not show negative effects on joy- and self-expression-related motivation, as argued by opponents. In contrast and counter-intuitively, joy seems to be positively influenced by SWP. Fourth, approaching the measurement of 'SWP presence' through different variables is a worthwhile approach to follow in the future, as the construct of a dichotomous SWP Law variable proved to be too coarse. Further research is needed, however, in finding and refining non-jurisdictional measures like patent pressure.

For policy makers, the recommendation is to proceed carefully in this complex and unexplored field, as there is currently no empirical support for either the proponents' nor the opponents' position. The results from this study suggest that US-based FOSS developers have no advantages through the high presence of SWP. Therefore, if the EU is planning to introduce a similar SWP policy (as it has tried in the past), it should provide compelling evidence of how the FOSS community's motivation – an important source for its performance – would be positively affected.

As with most initial studies in a new field, there are some limitations to be acknowledged, which should be addressed in future research. First, measuring the software patent situation in a systematic, quantitative way is a considerable challenge. The coarse measure for SWP law applied in this study have most probably obscured some geographical and/or cultural effects, leading to biased results. Second, much effort was made to produce a sample of alive FOSS projects that would respond to a survey call. Yet, this effort in itself has led to a sample bias. A future study looking specifically at why dead FOSS projects actually died – or new ones were never born – would be an important addition to these results.

This study focused on developer motivation as one of the key ingredients for the FOSS system to function. A next challenge will be to investigate the effects of software patents on *innovative behavior* of FOSS developers because only if they foster innovation, should they be present in the FOSS system.

7. References

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

Allison, P.D. (2002). *Missing Data*. Thousand Oaks, CA: Sage Publications.

Bessen, J., Hunt, R. (2007). An Empirical Look at Software Patents. *Journal of Economics & Management Strategy*, (16)1, 157-189.

Bessen, J., Hunt, R. (2004). The Software Patent Experiment. In OECD (Ed.), *Patents, Innovation and Economic Performance* (pp. 247-263). Paris: OECD Publishing.

Bessen, J., Maskin, E. (2006). *Sequential Innovation, Patents, and Imitation*. Institute for Advanced Study, School of Social Science, Working Paper Economics No. 0025. 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.

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

Czaja, R., Blair, J. (1996). *Designing surveys : a guide to decisions and procedures*. Thousand Oaks, CA: Pine Forge Press.

Dapp, M., Bernauer, T. (2009). *Hot Debate about Chilling Effects : Do Software Patents Hamper Free/Open Source Software Development?*. Center for Comparative and International Studies, ETH Zurich, Working Paper 40/2009.

Dillman, D.A. (2000). *Mail and internet surveys: the tailored design method*. New York: Wiley.

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

English, R., Schweik, C.M. (2007). Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. *UPGRADE*, (IX)6, 54-59.

European Commission (2005). *Patentability of computer-implemented inventions*. Retrieved 24.04.2008 from http://ec.europa.eu/internal_market/indprop/comp/index_en.htm.

European Information and Communications Technology Industry Association (EICTA) (2000). *Response to the European Commission's Consultation Paper on "The Patentability of computer-iomplemented inventions"*. Retrieved 05.02.2009 from ec.europa.eu/internal_market/indprop/docs/comp/replies/eicta_en.pdf.

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

Foundation for a Free Information Infrastructure (2009). *Software Patents in Action*. Retrieved 13.03.2009 from http://eupat.ffii.org/patents/effects/.

Fowler, F.J. (2002). *Survey Research Methods*. Thousand Oaks, CA: Sage Publications.

Fowler, F.J. (1995). *Improving Survey Questions : Design and Evaluation*. Thousand Oaks, CA: Sage Publications.

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.

Ghosh, R.A., Aigrain, P. (2006). *Economic impact of open source software on innovation and the competitiveness of the ICT sector in the EU*. Retrieved 20.01.2007 from http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf.

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.

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).

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.

Howison, J., Conklin, M., Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, (1)3, 17-26.

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

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

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.

Lévěque, F., Ménière, Y. (2004). *The Economics of Patents and Copyright*. Berkeley: Berkeley Electronic Press.

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.

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.

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

Rentocchini, F., De Prato, G. (2006). Software Patents and Open Source Software in the European Union: Evidences of a Trade-Off?. In Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G. (Eds.), *Open Source Systems* (pp. 349-351). Boston: Springer.

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.

Royston, P. (2005). Multiple imputation of missing values: update. *The Stata Journal*, (5)2, 188-201.

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

Stallman, R.M. (2004). *Fighting Software Patents - Singly and Together*. Retrieved from http://www.gnu.org/philosophy/fighting-software-patents.html.

8. Appendix A – Descriptive statistics

8.1 Motivational factors

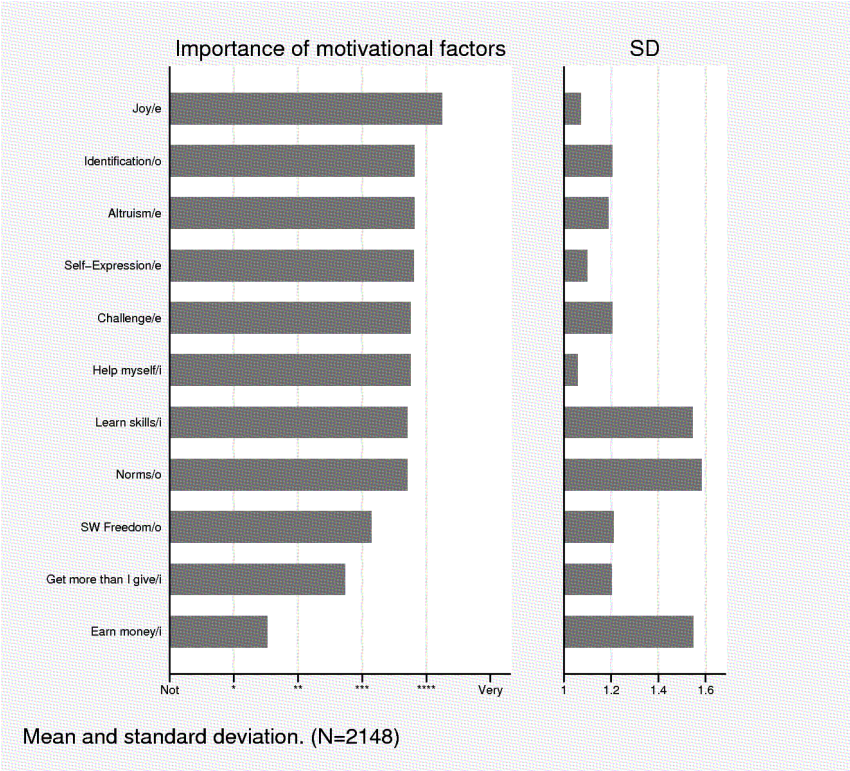

FIGURE-A 3: Importance of motivational factors

8.2 Experience

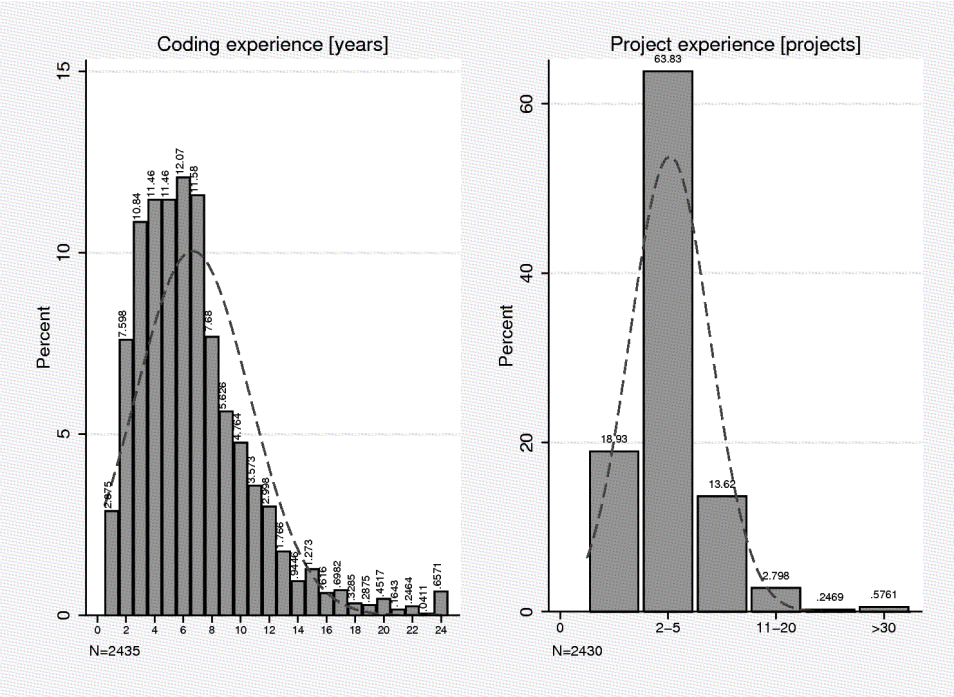

FIGURE-A 4: Respondents' coding and project experience

8.3 Education and age

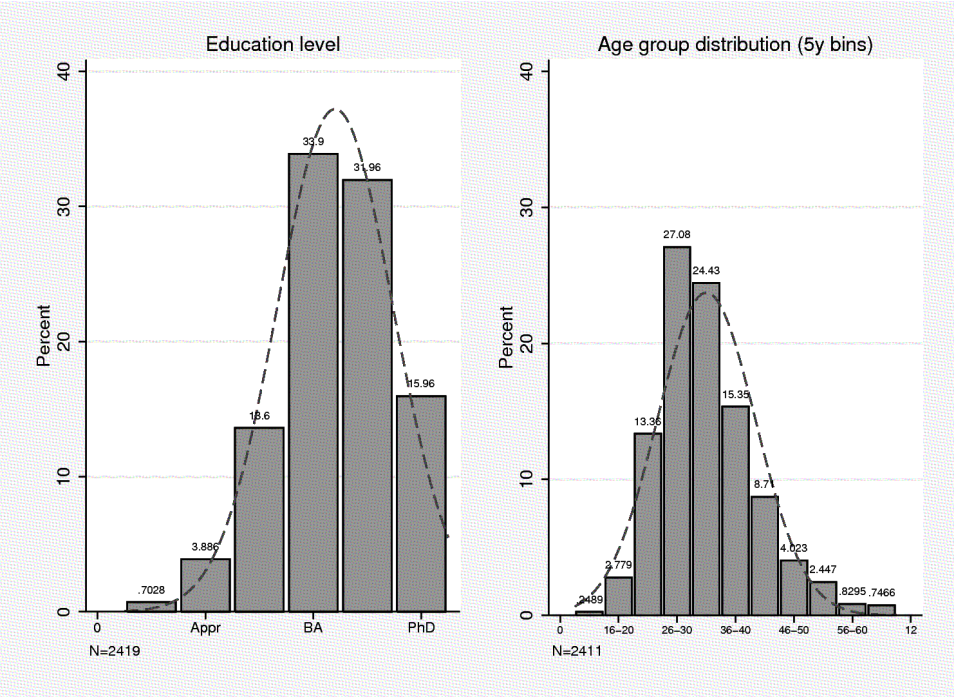

FIGURE-A 5: Respondents' level of education and age group

9. Appendix B – Underlying survey questions

Here, the questions and answer categories from the survey questionnaire used for this study are listed. The variable names used in the regression analysis are mentioned in bracketed capitals.

Motivation. Please rank how important the following motives are for you to contribute code to FLOSS projects!

	- - - Not at all	- -	-	+	+ +	+ + + Very	No answer
I simply enjoy programming. (JOY)	O	O	O	O	O	O	O
I want to create beautiful and elegant programs. (SELF-EXPRESSION)	O	O	O	O	O	O	O
I seek the challenge of solving programming tasks.	O	O	O	O	O	O	O
I feel good about helping others with my programs.	O	O	O	O	O	O	O
I value the goals of the FLOSS community.	O	O	O	O	O	O	O
I support the technical goals of my project.	O	O	O	O	O	O	O
I think software should be free. (SOFTWARE FREEDOM)	O	O	O	O	O	O	O
How my family/friends see my engagement is ...	O	O	O	O	O	O	O
I get back more than I contribute.	O	O	O	O	O	O	O
I develop my programming skills. (SKILLS)	O	O	O	O	O	O	O
I solve my own programming problems.	O	O	O	O	O	O	O
I earn money from it. (MONEY)	O	O	O	O	O	O	O
I build a reputation as a good developer.	O	O	O	O	O	O	O
I improve my future career perspectives.	O	O	O	O	O	O	O
I think it is fair to give back when you take.	O	O	O	O	O	O	O
I fight against proprietary software.	O	O	O	O	O	O	O

Software patent influence. According to your knowledge, which application domains are mostly affected by software patents? Select a maximum of four (4) domains!

O	Communications	O	Database	O	Desktop Environment
O	Education	O	Formats and Protocols	O	Games/Entertainment
O	Internet	O	Multimedia	O	Office/Business
O	Scientific/Engineering	O	Security	O	Software Development
O	System	O	Text Editors		

CHAPTER IV

Dances with patents – The role of motivation and software patents in the innovation behavior of FOSS developers

Preliminary results from this paper were presented at the OpenExpo conference in Berne (CH). I thank Bob English and Charles Schweik for providing project information for the sampling, and Google, OpenMoko, and ETH Zurich's Neptun program for sponsoring the lottery prizes for the survey.

Abstract

Free/Open Source software (FOSS) is investigated for its exceptional way to produce software in a collaborative, ownership-sharing, often non-commercial manner. Advocates consider FOSS to be very vulnerable towards software patenting because of its unorthodox setup. This article investigates how the motivational setup in connection with the presence of software patents affects individual innovation behavior of FOSS developers. It proposes a new metric for innovation behavior and an additional metric for software patent pressure. Hypotheses are tested against a new data-set from a large developer survey. Findings are that (1) intrinsic motivation leads to more innovative, and extrinsic motivation leads to more reuse-based code contributions; (2) software patent presence per se has no observable empirical effect on innovation behavior, be it positive or negative, except (3) for reverse-engineering contributions, which seem to correlate with software patent presence.

1. Introduction

The debate whether software patenting helps or hurts Free/Open Source software (FOSS) development is ongoing. Opponents express concerns about the restrictive consequences of a strong presence of software patents (Foundation for a Free Information Infrastructure 2009), while proponents do not see Free/Open Source software as a special case warranting exceptions (European Information and Communications Technology Industry Association 2000). This article investigates how the presence of software patents affects the innovation behavior of FOSS developers, while considering the particularities of the developers' motivational setup. A set of hypotheses is proposed and empirically tested against a new set of data from a large-N survey of FOSS developers. Methodologically, new metrics to measure individual innovation behavior and software patent (SWP) presence are proposed.

One may argue that in every field where patents exist, legal issues occur and lawsuits take place; they are just part of the game where one party has exclusive rights that can challenge or be challenged by other parties. Yet, it has been argued that the FOSS system is particularly vulnerable to this threat because of its open and collaborative approach to software innovation. Some FOSS principles, such as operating voluntarily and giving away software licenses, raise the question whether monetary incentive instruments such as patents are adequate for this environment. The motivational setup, especially non-monetary factors, has shown to be an important characteristic of the FOSS system (Krishnamurthy 2006).

How does the presence of SWP influence the FOSS system and its actors? Despite long and heavy debates about software patenting in the US and the EU, especially its effects on the FOSS system, policy-makers still have only little systematic research to go on when trying to make their decisions. The situation is furthermore dissatisfying as policies in the US and the EU have been diverging: Empirical data that would support the pro-patent course of the US is lacking; and FOSS developers in the EU are not reassured that the EPO practice of patenting software on a case-by-case basis is not clearly prevented.

Despite this real-world relevance, this specific question still represents a gap in extant literature because the fields on patent theories and FOSS motivation are still mostly isolated. Based on initial theoretical work connecting both fields (Dapp and Bernauer 2009), the article develops a theoretical argument and proposes a set of hypotheses that link SWP presence and important mo-

tivational factors to the innovation behavior of FOSS developers. For that, a new metric is presented to measure innovation behavior based on code distribution type. Also, different metrics are proposed to capture 'SWP presence'. Besides measuring SWP availability and incidents in a country, a patent pressure index based on software domains is proposed.

The article structure differs from the typical theory-hypotheses-operationalization flow. As the hypotheses make use of the proposed new theory-driven metrics, the new metrics are discussed with the theoretical arguments before deriving the hypotheses. To help the reader, the next section provides not only a brief literature review, but includes an overview of the new concepts making up the analytical framework for this study. The section after that discusses in detail the theoretical arguments for each concept and derives two sets of hypotheses for empirical testing: the first set connects motivation and innovation behavior, the second connects SWP presence and innovation behavior. The research design section describes the survey process and describes in detail the operationalization of variables. The results section describes the statistical results and provides interpretations. The final section gives a summary and offers recommendations to policy-makers and ideas for future research.

2. Literature review and analytical framework

This section gives a very brief introduction how the FOSS system works before presenting the three main concepts used in this study: innovation behavior and motivational setup of FOSS developers, and the presence of software patents. It concludes with an analytical description of how these concepts relate to each other in order to prepare the reader for the details of the theoretical argument presented in the following section.

Free/Open Source software (FOSS) such as Linux is software that is freely available on the Internet for everyone to use and contribute. It is provided by a large number of developers who may be volunteers, but who may also be hired by companies to contribute. Thus, FOSS is produced and distributed under much different conditions compared to traditional, so-called proprietary software, which is based on registered license agreements, such as products by Microsoft.

This massive, often anonymous, online collaboration to produce software has been made possible since the Internet is widely available and personal computers have become comparably cheap. FOSS development is organized in projects. A project can be compared to an association that an interested developer can join. The developers communicate, coordinate their work, and share code fragments using web portals in the Internet such as SourceForge (www.sourceforge.net) or own dedicated websites if the project is large (e.g., www.openoffice.org). From an organizational perspective, different relations have developed and co-exist today: they can range from an individual hobbyist developer who spends his spare time contributing code to a project of his choice to more recent setups in which employees are hired by a company to work on a FOSS project because it forms part of that company's offering to clients. A more detailed introduction to the FOSS system and its particularities can be found in Dapp and Bernauer (2009).

Measuring innovation behavior. The traditional, manufacturing-centric distinction between invention, innovation, and diffusion (OECD 1997:9) does not adequately capture the service sector and its intangible output (Smith 2005:169), which includes FOSS too, as the development process is closer to delivering a service than a product. Also, a price-based market for FOSS code like the market for proprietary software products does not exist. Money is paid for services around the code and sometimes for customized software development. A third aspect is that innovation is usually understood as something happening in a company context, as many contributions in the *Oxford Handbook on Innovation* illustrate (Fagerberg 2005). Yet, the FOSS community creates new

and useful software in a context characterized by ad-hoc organization, volunteering, and shared ownership of code. O'Mahony (2005) identifies parallels to common pool resource management, and other scholars have described it as a »private-collective model of innovation« (von Hippel and von Krogh 2003a) or »commons-based peer production« (Benkler 2002). FOSS innovation resembles the academic way of sharing and building upon the results of others (Lerner and Tirole 2004:31), and Saviotti's evolutionary model of innovation (Marinova and Phillimore 2003:49) adequately explains many aspects of the FOSS innovation system, although more research is needed here.

In the absence of the commercially driven distinction between an invention and an innovation offered on a market, only *newness* (cf. Rogers 2003:12) as the essence of innovation remains to characterize FOSS innovation. According to Bessen and Maskin (2006:2), FOSS development offers two paths to how innovation occurs: sequential and complementary. 'Sequential' means that each successive invention builds on the preceding one, while 'complementary' means that every innovator independently follows his own research line, which increases the overall probability to reach a particular innovation goal within a given time frame (Ibid.). Thus, 'newness' stems from two equally important sources. First, the *competition of independent ideas* – different, independent paths that can all lead to competing programs with equivalent functionality; second, the *imitation of existing ideas* – building on and extending existing programs. Blind *et al.* (2003:84) support this claim empirically by reporting a code reuse share of 30% among German software companies compared to a 70% share among independent developers in the FOSS community. This aspect is relevant for the discussion about copyright and patent protection of software (see next section) because both protection mechanisms foster the first and hinder the second source of innovation in FOSS. Finally, FOSS contests the notion that organizations are the locus of innovation. Watts (2003), for example, argues that the final sources of new inventions are very often traced back to individuals, while the transformation into market innovations is done by networks.

The proprietary software industry usually applies a user-centric innovation perspective: "new" meaning 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). Wheeler argues for measuring innovation on the developer side. He also claims that most FOSS development consists of recombination or integration of existing components, just as in the proprietary software industry – and is thus not innovative. He proposes a definition under which only *new programming paradigms* qualify as innovations. Yet, this very demanding definition allows for only a few innovations per decade and is not practical for most FOSS studies spanning a few years at most. In the next section, a developer-centric, but less demanding, metric is presented.

Motivational setup. 'FOSS' as a mechanism of having large groups of volunteers contribute to a public good useful to many, where free-riding by non-contributors is encouraged rather than sanctioned, has raised the interest of scientists. The question why developers engage in FOSS without a direct monetary reward was one of the first tackled (Lerner and Tirole 2001; von Hippel and von Krogh 2003b). Motivational factors are divided into extrinsic and intrinsic factors. If a developer feels motivated to contribute because he finds it – for different reasons – inherently enjoyable for himself, such motivation is called 'intrinsic' (Ryan and Deci 2000). Intrinsic factors can be divided into enjoyment- and obligation-based factors. Enjoyment-based factors are related to the single individual, while obligation-based factors relate to the larger community. Extrinsic factors, on the other hand, motivate by providing additional and separate benefits such as money or new skills. They are further grouped into a) factors materializing instantly and thus acting as rewards and b) factors that come with a delay, thus acting as incentives.

From the broad spectrum of possible motivations (Krishnamurthy 2006), Dapp and Bernauer (2009, Table 4) identified a list they argue as potentially affected by software patents (cf. TABLE 2 in the next section). The next section will discuss in more detail which factors have been used for the study and why.

Software patent presence. In order to understand the issues of patenting software, particularly FOSS code, it is important to understand the differences between copyright and patents. Copyright regulates the make, use, sale and other distribution of exact copies of a work or parts thereof. Copyright protects the form of a work, not the underlying idea or concept described. For example, Albert Einstein is the author of several books on the theory of relativity, yet copyright allows everyone to also write books on the subject – as long as no text from the original author is copied without permission.

Software, whether proprietary or FOSS, is subject to copyright law (beginning 1973 in the US, gradually introduced in other countries; since 1991 regulated on EU level). Today's business

model in the software industry is mainly based on the exclusive rights given by copyright law: users buy licenses to obtain the right to *use* the software. Other activities, such as copying, modifying or distributing are not allowed unless the software vendor allows it ("All rights reserved"). As only the vendor owns and controls the code, this type of software is called "proprietary". Producing similar software independently is allowed, however, since copyright law only protects the form and not the underlying ideas. Thus, different programs for similar tasks are available, and competition is undistorted. Dapp and Bernauer (2009) argue, however, that technical attributes of software help monopolistic structures to arise more easily than in other industries.

Although FOSS is covered by copyright, the FOSS community applies copyright law with a very different goal compared to proprietary software vendors. To make sure that the access and sharing of code – the key activities making the FOSS system work – are not hampered, *copyright licenses* have been developed that legally enshrine the rights to run, modify, copy, and distribute code by anybody. This constitutes the legal basis for intensive combining and reusing of code inside the FOSS community.

Patents, on the other hand, protect the underlying concept, thus covering all possible forms of implementation. A patent is a set of exclusive rights granted by the state to an inventor for a limited period of time (Scotchmer 2004, ch. 3) to prevent others from making, using, selling, offering for sale, or importing the patented invention. A new concept or 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 ('three-step-test'). Once a patent is granted, all implementations 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. See Dapp and Bernauer (2009) for a more detailed discussion related to patenting of software.

The presence of software patents is most directly determined by the patent law of a jurisdiction. If the patenting of software is possible, software patents may be present. If the patent law does not allow patenting of software, they should not be present. For two reasons, the situation is not as clear-cut as one would expect: first, patent law itself can leave room for debate. Article 52(1) of the European Patent Convention (EPC) states that 'programs for computers' are not inventions, whereas article 27(1) of the Trade-Related Aspects of Intellectual Property Rights (TRIPS) annex to the WTO agreement states that "patents should be available (...) in all fields of technology" and the proposed, and rejected, EU directive on computer-implemented inventions based patentability on the construct of a 'technical' distinction between technical software and

software 'as such'.[25] This led to the interesting discussion what technical software is and how it differs from software as such. The US, on the other side, is less vague as several court decisions have lowered the barrier to obtain (and defend) software patents over the last decades (Cohen and Lemley 2001), which led to what Bessen and Hunt (2004) called the 'software patent experiment'.

This study introduces a second metric for SWP presence based on the patent pressure in a software domain. Certain domains of software, such as multimedia or file formats, can be more prone to software patents than others, because the players belong to industries more used to patenting (e.g., electronics or semiconductors) or because patents are considered an easy means to block competitors.

How do the three concepts – innovation behavior, motivational setup, and SWP presence – connect? The fundamental proposition put forward in this article is that FOSS developers choose their code contributions – thereby the level of their *innovation behavior* – in accordance to their *motivational setup,* while the *presence of software patents* can influence this choice. The general expectation is that higher levels of motivation lead to higher levels of innovation behavior. As to the effect of a stronger presence of software patents, arguments of SWP proponents and opponents – who predict opposite effects – are discussed and tested.

In this model, SWP presence and motivation are considered as independent factors. No empirical support for a direct causal relationship between these factors has been found in a study by Dapp (2009).

25 The original paragraph (European Commission 2002:6) reads: "Under Art. 52(2) of the EPC, programs for computers "as such" are defined as not being inventions and are thus excluded from patentability. The Boards of Appeal of the EPO have held that it is fundamental to all inventions that they have a technical character. Similarly, Article 27(1) of the TRIPS Agreement confirms that patents shall be available for inventions in all fields of technology. Accordingly, the EPO Boards of Appeal and courts of the Member States have held that computer-implemented inventions can be considered as patentable when they have a technical character, i.e. when they belong to a field of technology. Computer-implemented inventions which meet this condition are not considered to fall under the exclusion in Article 52(2) as they are considered not to relate to programs for computers "as such". In fact, the exclusion has been interpreted by the Boards of Appeal of the EPO as relating to those computer-implemented inventions which have no technical character."

3. Theoretical concepts, arguments, and hypotheses

This section has three aims: First, it discusses in detail the main theoretical concepts – innovation behavior, SWP presence, and motivational setup. Second, new metrics to measure innovation behavior at the individual level and SWP presence related to domain-specific patent pressure are proposed. Third, based on theoretical arguments and the proposed metrics, two sets of hypotheses are put forward: the first set links SWP presence and innovation behavior, the second set links motivational setup and innovation behavior.

3.1 From code contributions to individual innovation behavior

For the purpose of this study, FOSS innovation is defined on the level of the individual developer and his code contributions. A new metric is proposed as shown in TABLE 1: depending on the type of code contribution, a developer is considered more or less innovative. Contributions are thus arranged in three groups: writing algorithms, reusing code, and reverse-engineering code. While writing algorithms is considered to be more innovative than reusing existing code, reverse-engineering is not part of the ordinal scale because it is a very specific way of creating code.[26]

TABLE 1: Measuring innovation behavior of individual code contributions

	Type of code contribution	Category	Variable
1	Inventing new algorithms/methods to be implemented in code	algorithm-based	AL2
	Coding of known algorithms/methods (e.g., from literature) from scratch		AL1
2	Recombining existing FOSS components with much adaption	reuse-based	RU2
	Integrating existing FOSS components with little adaption		RU1
	Linking to existing FOSS libraries		LIB
3	Reverse-engineering, i.e., imitating functionality from non-FOSS programs	-	REV

The first category encompasses implementing *algorithms,* the 'mathematical recipes' for solving problems with a computer. Devising a new algorithm for a problem is considered the highest level of innovation behavior (variable AL2). This means the developer invents a new algorithm/method to solve a problem and implements it in source code. Example: Devising a new way of scheduling trains in a country and writing software based on this new algorithm. Translating an existing algorithm into a computer program is next on the scale (AL1). This means the developer implements an algorithm that he learned or read about in a textbook, without access to

26 »Bug-fixing« (i.e., correcting defective code) is not considered for the purpose of this study.

other programs solving the same or similar problems. Example: An audio engineer needs a program capable of performing Fourier-transformations on his wave data. He transforms the algorithm described in mathematical terms in a book into software code using a programming language. Depending on his expertise, the program can be faster of more efficient than other programs performing the same.

The second category encompasses *code reuse.* Three forms are distinguished: Recombining existing components from other FOSS projects with a considerable amount of adaption (RU2) is considered to be more innovative than simply integrating existing components with only little adaption to make them work together (RU1). The lowest level of innovation is "linking libraries" (LIB), when developers include library functionality in own programs. Example: A graphics program needs to be able to draw rectangles with a mouse. This functionality has been implemented in various programs and may even be ready for use by linking to one of the graphics libraries available.

The third category is not rated within the innovation scale as it denotes a very specific way of writing code: *Reverse engineering* is "the process of extracting know-how or knowledge from a human-made artifact" (Samuelson and Scotchmer 2002:1577). Reverse engineering (variable REV) is used in cases where the object code form of a program can be used (the program can be "run"), but no access to the source code form is available, thus no modifications are possible. Technically, it means to disassemble and decompile the machine-readable object code into human-readable source code with the help of specialized software tools. From the resulting code fragments inferences can be drawn of how the original software was programmed. This original functionality is then reproduced by writing a new program based on the extracted code fragments. Thus, 'reverse engineering' reverses the default process of software development: a developer writing source code that is translated into object code.

How innovative is reverse engineering? Usually, reverse engineering is not done to be innovative but for the purpose of interoperability – to make two programs interact with each other and/or exchange data better. Therefore, it is not 'competition of independent ideas' since existing software is replicated. Yet, it is not simple imitation either because comparable efforts are required to reverse-engineer software as are needed to write it in the first place (Mitchell 2005:27). Free-riders avoid such efforts when making (identical) copies, whereas developers go through the whole development process when reverse-engineering. Example: To be able to inter-operate and

exchange documents, the developers of the OpenOffice.org project reverse-engineered the document format used by Microsoft Office, which is not publicly documented.

The metric from TABLE 1 is used to *measure the innovation behavior* of code contributing individuals. The three groups of different code contributions are represented by the following variables on a 6-point ordinal scale. For the group with the highest innovation level, *algorithm-based code contributions*, two variables distinguish whether the developer implements new (AL2) or known (AL1) algorithms in code. For the next category, *reuse-based code contributions*, RU2 denotes reuse by recombining with much manual adaption by the developer and RU1 denotes reuse by integrating with only little manual adaption. The lowest level, *linking to libraries* to access their functionality, is measured by LIB. The last variable, REV, is outside the innovation scale as it measures code contributions based on reverse engineering.

3.2 How motivational setup affects innovation behavior

Dapp and Bernauer (2009) identified a set of motivational factors of FOSS developers, shown in TABLE 2, that is potentially affected by the presence of software patents.

TABLE 2: Motivational factors potentially affected by SWP

Instant Rewards (extrinsic)	Enjoyment-based (intrinsic)
SKILLS (*). Learn new coding skills by reading and writing code.	JOY (*). Programming as a fun activity, like other hobbies.
SELF-HELP. Ability to help oneself by improving programs.	SELF-EXPRESSION (*). Ability to express oneself aesthetically through software code. Code writing as an art form.
NET GAIN. A developer gets access to the whole program, although he contributes only small parts.	ALTRUISM. Provide freely available useful software to support other users.
MONEY (*)(M). Direct monetary reward, e.g., when being hired to write code for a FOSS project.	CHALLENGE. The intellectual challenge of solving difficult programming problems.
Delayed Incentives (extrinsic)	**Obligation-based (intrinsic)**
CAREER (M). Signal skills to potential future employers, self-marketing.	IDENTIFICATION. Identification and belonging to a community.
	NORMS. Observance of community norms like, e.g., sharing.
	SOFTWARE FREEDOM (*). ‘Software must be free’ as a political mission.

*Base*d on Dapp and Bernauer (2009)

The following five factors (*) have been selected from TABLE 2 because of their overall importance and their relationship to software patents as well as to be comparable with the study of Dapp (2009) about the effects of SWP presence on motivation.

From the group of extrinsic factors, SKILLS and MONEY are selected as they relate directly to two important theoretical justifications of software patents: while MONEY relates to incentive theory, SKILLS relates to exchange theory (see next sub section for details on patent theories). From the group of intrinsic factors, JOY is selected because it is reported in this study as the overall most important motivational factor (FIGURE-A 1 in the appendix shows the importance of all factors). SELF-EXPRESSION is the factor most directly affected by the restrictions SWP impose on code writing. A SWP reduces the number of options of how to formulate code because some will be covered by the SWP and hence be an infringement if used. The more SWP are present, the less free choice FOSS developers have to express themselves through code. From the group of obligation-based factors, SOFTWARE FREEDOM is of specific interest in the argument related to reverse-engineering-based code contributions. The level of this motivation may also be influential in a developer's decision to stick with a project or withdraw his engagement when facing a strong presence of software patents. These arguments will become clearer as the hypotheses are introduced below.

Motivational setup and code contributions. FOSS developers are driven by intrinsic and extrinsic motivation and they engage in different types of code contributions. Intrinsic factors like joy and self-expression are easier to trigger by programming tasks that are challenging and require the creation of new solutions than by tasks that can be done by reusing and combining existing code. It is more interesting and more fun to take up a challenge. Dealing with new code also accommodates the developers need to express themselves (in code) better because it gives them the opportunity to write code they way they prefer – not unlike artists who are allergic towards constraints. Good code in the eyes of FOSS developers is not only functional, but also aesthetic or elegant. Consequently, such original contributions are highly regarded in the community. As developers build reputation based on contributions their focus will be on challenging algorithms rather than on code reuse – if they can choose freely.

On the other hand, extrinsic motivational factors like money and learning skills may trigger reuse-based rather than algorithm-based contributions. Many FOSS projects suffer from the 'incomplete-last-mile' phenomenon: Most technically challenging tasks are already completed, what

is left are the less interesting tasks to make a program more user-friendly, et cetera. Tasks that are often repetitive across programs and usually include more code reuse than new code development. Developers who are eager to learn programming skills may start here; read existing code and learn recombining and adapting it before suggesting own modifications or improvements to the community. In other words, learning skills and reuse-based contributions should occur together. Ease-of-use and polished user interfaces are often considered very important from a user perspective. Therefore, users or customers (in the case of companies offering FOSS solutions) may offer payment to developers to complete the 'last mile' since intrinsic motivation alone may not be enough to get the job done. Developers motivated by money may indeed be attracted to this type of setup. In contrast, the paying party has strong influence on the developers' contribution, which may reduce the self-expression-related motivation. Combining all these arguments leads to the following two hypotheses:

HYPOTHESIS 1: FOSS developers with above-average joy- and self-expression-related (intrinsic) motivation are more likely to contribute algorithm-based code. FOSS developers with above average monetary and skills-related (extrinsic) motivation are less likely to contribute algorithm-based code.

HYPOTHESIS 2: FOSS developers with above-average monetary and skills-related (extrinsic) motivation are more likely to contribute reuse-based code. Developers with above-average joy- and self-expression-related (intrinsic) motivation are less likely to contribute reuse-based code.

Reverse engineering is a different situation. It is mostly carried out to create a free, open source alternative that is functionally equivalent or capable of inter-operating with a proprietary program, with the goal of reducing dependence on that program. Developers engaging in such projects are usually strongly motivated by the 'free software' philosophy – of having as much software as possible free (e.g., the Free Software Foundation regularly calls for support to create replacements for targeted proprietary programs). Hence, high rates of reverse engineering should go together with high levels of software freedom-related motivation.

HYPOTHESIS 3: FOSS developers with above-average software-freedom-related motivation are more likely to contribute reverse-engineering-based code.

3.3 How software patent presence affects innovation behavior

Incentive theory, reward theory, and exchange theory are the classical utilitarian justifications for patents as instruments to foster innovation (Fisher 2005). The most prominent theory, *incentive theory*, argues that patents enhance innovation because they give an incentive to the individual to invent and commercialize the invention. Patents generate the necessary up-front investments needed for development (Mazzoleni and Nelson 2004). *Reward theory* argues that patents "secure inventors their just reward, proportional to the usefulness of the invention to society" (Fisher 2005:8). The exclusive right to commercialize the invention is justified because its potential to perform tasks more efficiently or satisfy needs more effectively represents a cost saver to users that is measurable in monetary terms. Patents increase the profit of the inventor and discourage competitors from free-riding. *Exchange theory* argues that patents – because of the disclosure requirement – offer a fair balance between the interests of the public and the inventor. Through the patent letter, the public gets access to the knowledge embedded in the invention, while the inventor gets exclusive rights in exchange for disclosing the invention. Through the construction of a time-bound exclusive right that balances the interests of the inventor and the public, patents encourage both the invention and the making public of it.

On the other hand, critics claim patents to be unnecessary or even detrimental to innovation. 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 commercial incentive would still be sufficient even without patents (Boldrin and Levine 2008). Specific to software, Bessen and Maskin (2006) have argued that patents are detrimental for markets with sequential innovation like the software industry. Klemens (2005) raised the question whether software code is a patentable subject matter at all as algorithms are essentially math; and math is not, according to traditional justification and logic, patentable.

Empirical research about the innovation effect of patents is not conclusive. Patents appear to be most effective in the drugs, chemical, and biotech industries, but seem to have very little innovation effects in other industries (Cohen *et al.* 2000; Arora *et al.* 2003; Sakakibara and Branstetter 2004) – particularly compared to other strategies to commercially exploit inventions such as lead time or secrecy (Sattler 2003; Arundel 2001; Harabi 1995). Whether the negative or the positive effects of patents prevail in the software industry is a particularly heavy debate. See Dapp and Bernauer (2009) for a summary of the arguments.

Based on the concept that 'software patents' are inventions embodied in software (Allison and Lemley 2000), I use the definition by Bessen and Hunt (2004:8): software patents cover a "logic algorithm for processing data that is implemented via stored instructions; that is, the logic is not hard-wired".

SWP presence is defined based on two concepts: SWP availability relates to the legal question whether patents on software can be obtained in a certain jurisdiction and will be measured by one variable; SWP prevalence relates to the question of how strongly SWP are present in such a jurisdiction and will be measured with two variables.

SWP availability looks at patent law to determine whether SWP are at all legally available in a jurisdiction. It is only in such jurisdictions that SWP can be granted to patent holders and patent infringement can occur. Unfortunately, comparing patent law across different jurisdictions is a very complex process and dedicated legal studies on a per-country basis would be required to come up with a comparative index that captures the intricacies of patent law across countries – and it is uncertain whether it would be possible to come up with a simple-to-use index. To solve this problem, a simplified dichotomous classification for the variable SWP availability is used (cf. sub section on key variables). Levine/Saunders (2004) and Bessen/Hunt (2007) provide support for such an assignment.

SWP prevalence is measured by two different approaches. (a) As patent holders can defend their patents, patent infringements may lead to legal incidents (e.g., cease-and-desist letters, law suits). (b) The second measure is constructed orthogonal to jurisdictional borders. Anecdotal evidence suggests that certain software domains, like cryptography or multimedia, face higher SWP prevalence ('patent density') than other domains. A patent pressure index was constructed (cf. FIGURE 3) based on estimates given by respondents in the survey. From the list of 14 domain categories used on SF, respondents were asked to select the top four with the highest perceived patent pressure according to their knowledge and assessment.

How does SWP presence influence individual code contributions and hence innovation behavior? Based on Dapp (2009), the arguments put forward by SWP proponents and opponents can be summarized as follows:

SWP opponents argue that the presence of SWP decreases innovative activity and leads to less algorithm-based code contributions because SWP presence negatively affects joy- and self-ex-

pression-based motivation (cf. hypothesis 1). A developer who contributes mainly for these two reasons is not willing to face a lot of legal risk from SWP for making his contributions public – particularly if it is not a commercial activity for him. Furthermore, as only very few of the broad spectrum of motivational factors are monetary, the presence of SWP does not present an additional incentive for FOSS developers to engage in highly innovative contributions; it is rather the opposite, it has a negative effect because it restricts developers in freely expressing their new ideas in code (cf. hypothesis 2).

HYPOTHESIS 4 (opponents): Stronger SWP presence reduces the odds for above-average algorithm-based code contributions by FOSS developers.

On the other hand, SWP proponents argue that the presence of SWP increases innovative activity and leads to more algorithm-based code contributions. This is because SWP provide a means of compensation and an incentive to write new code since developers cannot earn money with selling FOSS code. Furthermore, the disclosure of the invention to the public through a patent reveals knowledge about the invention on a conceptual level that is more appropriate than low-level source code. Both mechanisms, triggered by the presence of SWP, lead to a higher potential for new and innovative code contributions.

HYPOTHESIS 5 (proponents): Stronger SWP presence increases the odds for above-average algorithm-based code contributions by FOSS developers.

How does SWP presence affect reverse-engineering? The argument of hypothesis 3 can be extended, as the software-freedom-related motivation does not only refer to software code but to the patent situation as well. A proprietary program in a software domain that is additionally covered by SWP calls out even more for a 'free solution' that is not only free in the sense of copyright ("FOSS") but also free in the sense of patent claims. For this reason, the presence of SWP in a software domain may *additionally* trigger reverse-engineering code contributions.

HYPOTHESIS 6: Stronger SWP presence increases the odds for above-average reverse-engineering-based code contributions by FOSS developers.

4. Research design

4.1 Data collection, sampling strategy, and survey design

Data collection. Data about innovation behavior, motivational setup and (partly) SWP presence was collected using an online survey targeted at leaders of FOSS projects registered on SourceForge.net (SF). SF hosts a broad set of projects from different software domains with many developers dispersed across different jurisdictions. Likewise, their employment status is expected to vary more than the population of a single large FOSS project such as the Linux kernel.

Sampling strategy. The typical difficulty with SF – to distinguish between abandoned ('dead') and alive projects – was mitigated with support from English and Schweik (2007). They developed a metric to separate successful projects from failed ones using release numbers and time intervals between releases, and kindly provided me with a list of 57,085 names of 'alive' projects that had produced at least one code release by August 2006. Project and developer information was extracted from the FLOSSmole database project (Howison *et al.* 2006), whose aim is to collect data about many FOSS projects for academic research. The sample frame included project leaders of all 57,085 'alive' projects hosted on SF as of August 2006. From these a random sample of 11,000 was drawn and invited to participate in the web-based survey. In total, 2,441 individuals responded (22% overall response rate), of which 1,815 resided in the US or the EU. The latter form the basis of this study.

Sampling bias. An intended selection bias is contained in the sample, as 'failed' projects are not considered in the survey since abandoned projects are much less likely to respond and provide reliable answers in a survey setup. As a consequence, the sample does not include developers who abandoned projects or who never got involved in a project because of SWP issues outside the 2-year period of the survey. An ideal sample would of course include past, current, and future FOSS developers. That being impossible for practical reasons, the question is in what way does this selection effect bias the sample and the results? I submit that the bias should not have systematic selection effects on the dependent variables, i.e., the code contribution types. There is an unintended bias on the predictor side, however, because motivation levels are most likely to be higher in the sample than in the population, as only 'motivated' and still actively engaged developers had a chance to participate in the survey. Vice versa, developers with low motivation levels are underrepresented in the sample. Since the bias happens on the predictor side, the empirical

results are biased accordingly, but the general relationship argued for in the hypotheses is not affected as the dependent variables are not affected by the sampling bias.

Survey design. A questionnaire was designed and tested in two pilot phases – a lab pilot and a field pilot – preceding the main survey run (Fowler 2002; Fowler 1995; Dillman 2000; Czaja and Blair 1996). The lab pilot was used to get feedback from 20 researchers and FOSS developers on the online setup as well as on wording used. The field pilot with a sub sample of 1% of the target sample was run with a first questionnaire: this pilot collected answers on a few open questions that were clustered and recoded into multiple-choice questions for the main run.[27] In autumn 2007, the main survey was run in three parallel batches of one-week intervals to reduce the risk of a technical failure stopping the survey process. Over four weeks, respondents in every batch received four emails: one invitation, two reminders, and a last call. Consequently, the main survey took six weeks to run.

A general challenge in surveys is to avoid sample distortion by biased respondents. The main cause of such distortion is self-selection of participants. To avoid the practice used by other developer surveys of recruiting participants through a 'snow-ball'-like system where the set of potential respondents is not defined ex ante, a controlled survey environment was set up, with a pre-defined sample frame, simple random sampling, targeted personal invitations, and mechanisms to prevent non-invitees from participating using the project data by FLOSSmole and English/Schweik.

4.2 Key variables

Dependent variables. For statistical analysis, the 6-point ordinal code contribution variables are dichotomized at their means to control for individual perception effects: not all respondents think of 3 on a 6-point scale as the same level of code contribution. This leads to a 1/0-scaling of the variables, where '1' denotes an above-average and '0' a below-average frequency in the respective code contribution type compared to the sample average.

In the predictor set for *motivational setup*, the importance of each motivational factor is measured on a 6-point Likert scale ranging from 'not important at all' to 'very important'. The following factors are incorporated in the analysis: MONEY, SKILLS, JOY, SELF-EXPRESSION, and SOFTWARE FREEDOM. To balance out self-rating effects, they have been dichotomized for the

27 Only answers from the field pilot (response rate 24.5%) were included in the final data set. Data from the lab pilot was not used.

analysis. Thus, '1' means an above-average level in that motivation factor, while '0' means below-average; always compared to the sample. (FIGURE-A 1 in the appendix shows descriptive statistics for all motivational factors.)

The predictor set for *SWP presence* consists of three variables. Legal SWP availability in a jurisdiction is captured in SWP LAW. The availability of SWP is coded as '1', the unavailability of SWP as '0'. This means respondents residing in the US were assigned a '1', respondents residing in the EU a '0'. Whether a FOSS project faced such an incident or not is captured in the dummy variable INCIDENT. The newly built patent pressure index of the 14 software domains used by SF has been categorized using a 3-point ordinal scale for the empirical analysis: high, low, and no domain pressure, with according variables HI DOM PRESS and LOW DOM PRESS included in the analysis. 'No pressure' serves as reference category and is left out to avoid model over-specification.)

Control variables. The following variables are included to control for effects outside the theoretical scope discussed: (a) SWP RESEARCH indicates whether or not a project researched the patent situation in its field, for instance using a patent database. (b) Whether a developer has direct (by coding) or indirect (other FOSS-based work) FOSS income is captured in the dichotomous variable FOSS INCOME. (c) HIGHER EDU captures whether or not a respondent has an advanced education degree (bachelor or higher). (d) Coding time is the time spent per week on FOSS development. CODTIME>5 flags if the developer spent more than 5h per week on FOSS projects, which is the median category of the sample population. (e) Experience with FOSS projects is measured in terms of number of projects (PJEXP). PJEXP>5 equals '1' if the developer has contributed to at least 5 FOSS projects. (f) Age information is represented in a simplified way by cutting the age-group curve at the median category: AGE>30 flags when a respondent is above 30 years old. Gender differences and project experience in terms of years were insignificant in the analyses and have been omitted.

5. Empirical results

5.1 Descriptive statistics

This section presents descriptive statistics for the dependent variables (five types of code contributions), the predictor variables used to measure motivation (five factors) and SWP presence (law, patent pressure by domain, and SWP incidents), plus some background information about the survey population. The list of relevant questions used in the survey to collect this data can be found in the appendix.

Population. The survey asked participants to provide information about the 2-year period August 2005 to August 2007. 637 (26,1%) respondents resided in the US, 1,178 (48,3%) in the EU, and 626 (25,7%) in the 'rest of the world' (ROW), yielding a total of 2,441 respondents (response rate 22%). The age groups 26-30 and 31-35 make up 51.5% of the sample – 16.4% are younger and 32.1% older. The large majority of respondents are male (1.5% females) and well educated (81.8% with a university degree), which is in line with the 'FLOSS' study that reported 70% developers with a university degree and a ratio of 1.1% females (Ghosh *et al.* 2002).

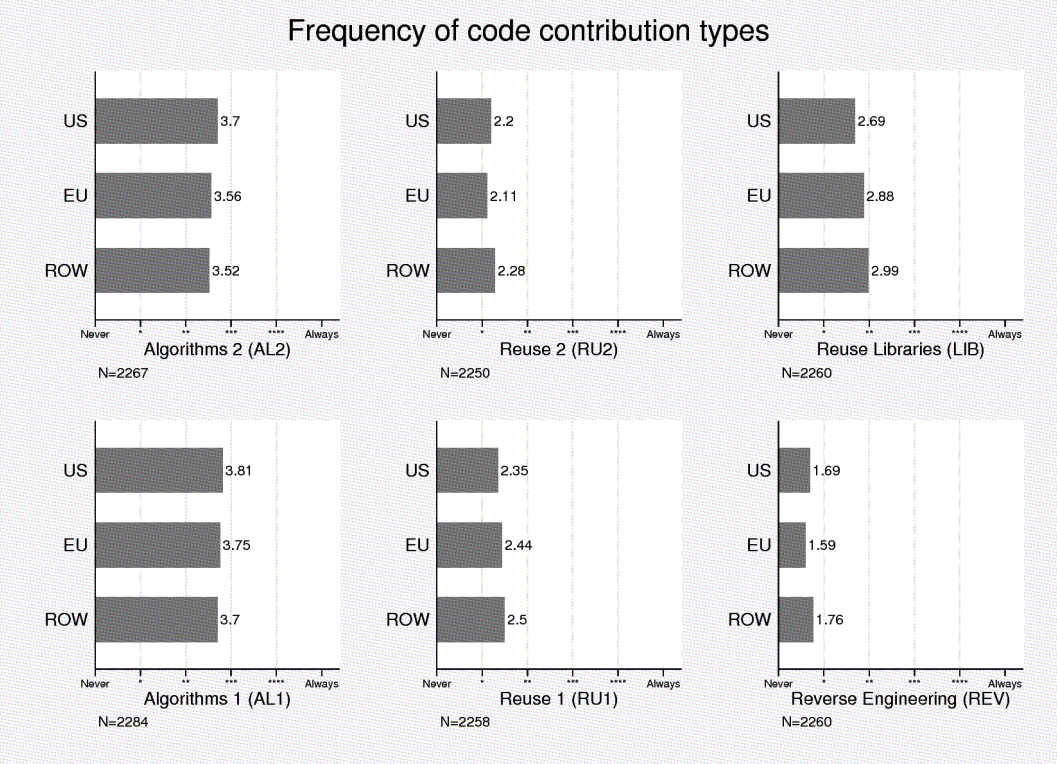

FIGURE 1: *Reported frequency of different code contributions (variable names in brackets)*

Dependent variables. FIGURE 1 shows frequencies of the different types of code contributions with the respective variable name given in brackets. One can see that developers in all regions reported more activity in the algorithm-based than in the reuse-based categories, except for reuse of libraries. The low prevalence of reverse-engineering was to be expected because only a small group of FOSS developers engages in this activity. Most developers appear to create sophisticated code – or at least they perceive it this way. A second observation is that the frequencies within a contribution type do not vary across geographic regions, indicating low variation in innovation behavior as well.

Predictor set 'motivation'. FIGURE 2 shows how important the different motivational factors are for respondents to contribute code. The overall picture for different regions is quite similar, with joy ranging as the top motivation and earning money at the end of the list. The middle field consists of self-expression, learning skills, and 'software freedom' – the motivation to support the philosophy of 'free software' as declared by the Free Software Foundation (2006). FIGURE-A 1 in the appendix gives a graphical overview of all motivation items used in the survey.

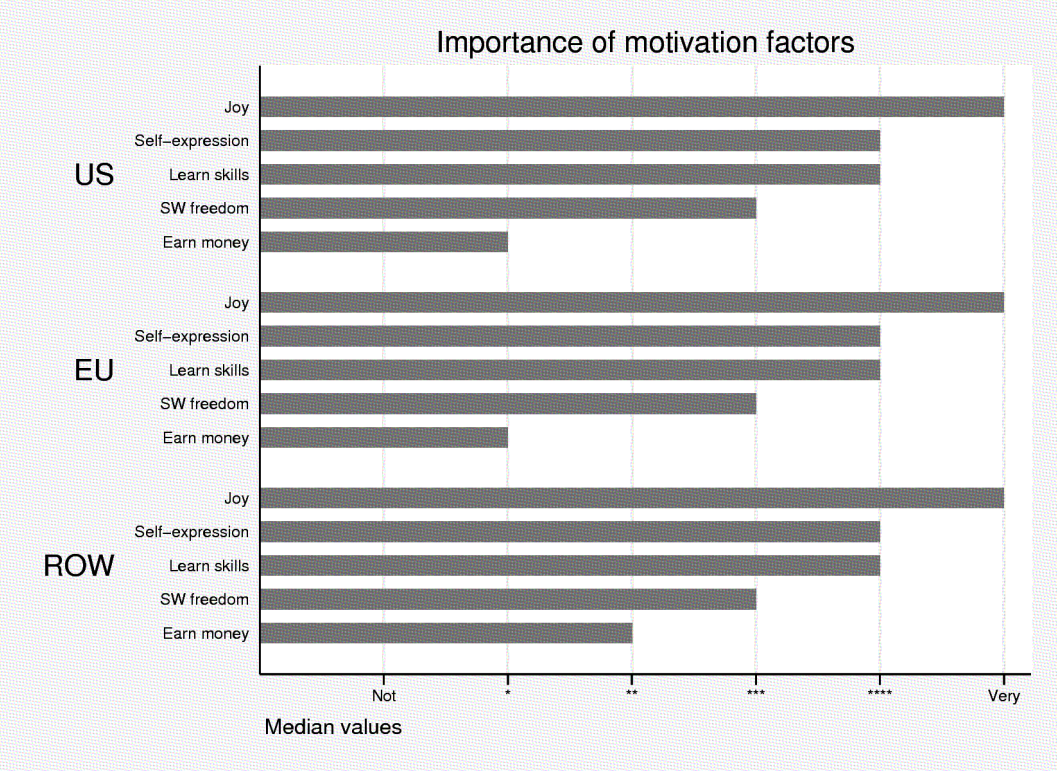

FIGURE 2: *Reported importance of different motivational factors*

Predictor set 'SWP presence'. Besides the geographic indication of SWP presence by jurisdiction (cf. FIGURE 1), a second measure was introduced – patent pressure related to software domains.

The top chart in FIGURE 3 shows to which software domain the sample projects belong. Some domains such as Software development (SWD), networking (NET), and games/entertainment (GME) are strongly represented, while others such as formats/protocols (FMP) or editors (EDT) are only weakly represented in the sample.

The bottom chart in FIGURE 3 shows the patent pressure index created from the assessments of the respondents concerning the patent situation in the different software domains. They were asked to name the four domains they expected to show the strongest presence of SWP. The top domains in this regard are multimedia (MIM) and formats/protocols (FMP), which is in line with anecdotal evidence that many SWP issues raised by the FOSS community often relate to these domains: formats *in* multimedia show a high patent pressure because many audio and video formats are covered by one or more patents; the most famous being the MP3 format. Most domains (from OFB to GME) range in the middle area, with education (EDU) and editors (EDT) showing very weak patent pressure.

To integrate this index information in the statistical analysis, two cut points were defined. They divide the patent pressure index into three ordinal categories: high (index >0.5), low (0.1<index<0.4) and 'no pressure' (index <0.1). High and low pressure categories are inserted in the specifications, 'no pressure' as the reference category is left out to avoid over-specification of the logistic model.

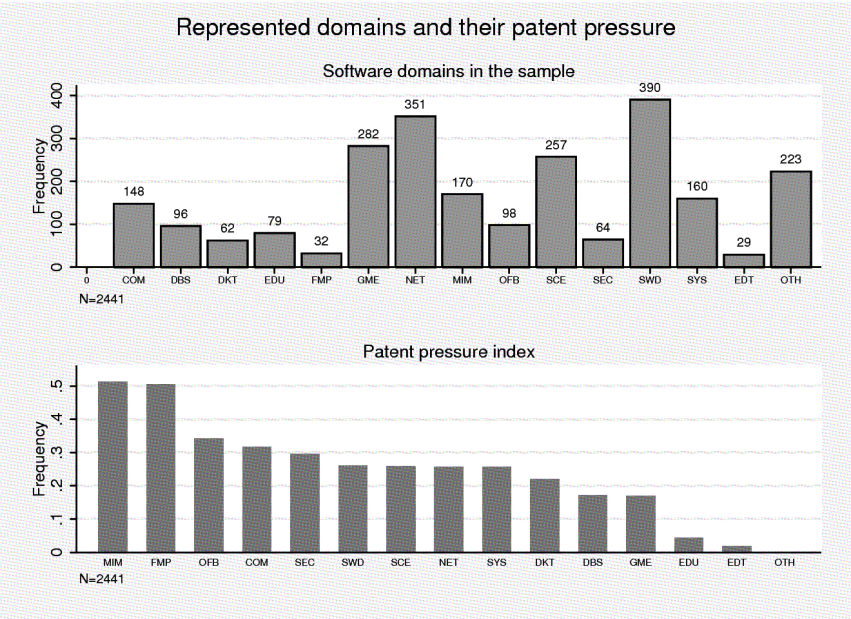

***FIGURE 3**: Software domains and patent pressure*

The third variable used to measure SWP presence indicates whether a project experienced an actual SWP incident – a patent holder approaching a project – during the reporting period. 44 out of 1,654 cases (respondents could skip the question) reported a SWP incident; that is 2.7%.

5.2 Regression analysis

Logistic regression was used to empirically model the effects of motivation and SWP presence on code contribution. Missing values on the dependent variables and predictors were imputed using multiple imputation procedures (Allison 2002) from Royston's package for Stata (Royston 2005). See part E of the main appendix for complete imputation statistics.

For each code contribution type, a logistic regression model was calculated with three different model specifications differing in the representation of the 'SWP presence' concept. The first specification includes the jurisdictional predictor (SWP LAW), the second patent pressure indicators (HI/LOW DOM PRESS), and the third SWP incidents (INCIDENT). Results are reported as odds ratios. The percentage figures given in brackets express the odds that a certain predictor triggers code contribution to 'jump' from below-average to above-average (i.e., from 0 to 1 in the logistic regression model).

As hypotheses make statements about more than one dependent variable at once, the presentation of the results is ordered by code contribution type. In other words, one table is used for more than one hypothesis.

The hypotheses related to motivation (H1-H3) are discussed below their respective table: TABLE 3 shows results for algorithm-based contribution types (AL2, AL1). Accordingly, hypothesis 1 is discussed directly below. TABLE 4 and TABLE 5 show results for reuse-based contribution types (RU2, RU1, LIB), with hypothesis 2 being discussed. TABLE 5 also shows results for reverse-engineering. Accordingly, hypothesis 3 is discussed directly below.

The hypotheses related to SWP presence (H4-H6) are jointly discussed after all tables have been shown because they make references to all code contribution types across all result tables.H1: TABLE 3 shows the effects of motivation (and SWP presence) on algorithm-based code contributions. It reports parameter estimates for H1, which contains two predictions about algorithm-based contributions: *FOSS developers with stronger joy- and self-expression-related (intrinsic) mo-*

tivation are more *likely to contribute algorithm-based code. FOSS developers with stronger monetary and skills-related (extrinsic) motivation are* less *likely to contribute algorithm-based code.*

TABLE 3: *Effects on algorithm-based code contributions (AL2, AL1)*

	Level 2 (AL2)			*Level 1 (AL1)*		
	A	*B*	*C*	*A*	*B*	*C*
SWP LAW	1.154			0.989		
	(0.132)			(0.113)		
HI DOM PRESS		0.686			1.038	
		(0.156)*			(0.238)	
LOW DOM PRESS		0.964			1.041	
		(0.148)			(0.160)	
SWP INCIDENT			0.562			0.827
			(0.262)			(0.371)
xM: MONEY	1.114	1.123	1.112	0.826	0.825	0.825
	(0.127)	(0.128)	(0.127)	(0.094)*	(0.094)*	(0.094)*
xM: SKILLS	0.741	0.739	0.741	1.085	1.083	1.085
	(0.086)***	(0.086)***	(0.086)***	(0.121)	(0.121)	(0.121)
iM: JOY	1.260	1.265	1.269	1.441	1.441	1.440
	(0.144)**	(0.144)**	(0.145)**	(0.169)***	(0.168)***	(0.168)***
iM: SELF-EXPRESSION	1.367	1.373	1.371	1.207	1.205	1.208
	(0.157)***	(0.158)***	(0.157)***	(0.144)	(0.143)	(0.144)
iM: SW FREEDOM	0.906	0.889	0.897	0.884	0.886	0.886
	(0.097)	(0.095)	(0.096)	(0.097)	(0.097)	(0.097)
SWP RESEARCH	1.589	1.657	1.655	1.853	1.854	1.870
	(0.283)***	(0.299)***	(0.298)***	(0.346)***	(0.347)***	(0.352)***
FOSS INCOME	0.942	0.932	0.952	1.126	1.125	1.128
	(0.129)	(0.128)	(0.131)	(0.156)	(0.156)	(0.156)
HIGHER EDU	1.065	1.072	1.060	0.980	0.980	0.980
	(0.147)	(0.148)	(0.146)	(0.135)	(0.136)	(0.136)
CODTIME >5	1.459	1.462	1.456	1.268	1.268	1.269
	(0.167)***	(0.168)***	(0.167)***	(0.149)**	(0.149)**	(0.149)**
PJEXP >5	0.942	0.943	0.937	0.782	0.783	0.783
	(0.137)	(0.138)	(0.136)	(0.109)*	(0.109)*	(0.109)*
AGE >30	1.047	1.056	1.070	1.202	1.201	1.201
	(0.114)	(0.114)	(0.115)	(0.135)	(0.134)	(0.133)*
aic	2215	2215	2215	2171	2173	2171
bic	2287	2293	2287	2242	2250	2242

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (e^{β}) with robust standard errors. N=1815.

Joy-related motivation increases the odds for above-average algorithm-based contributions by +26% (AL2) and +44% (AL1), respectively. For self-expression-related motivation the increase is only significant for AL2 (+37%). This result supports the first prediction of H1 concerning joy-related motivation, but gives only weak support concerning self-expression-related motivation.

Support for the second prediction (extrinsic motivation) is still weaker: Monetary motivation decreases the odds for above-average algorithm-based contributions only in the case of AL1 (-17%), while skill-related motivation decreases them only for AL2 (-26%). Of the control variables, patent research (plus 58-87%) and above-average time investment (plus 27-46%) both increase the odds for algorithm-based contributions.

Altogether, the results indicate that extrinsic and intrinsic motivations affect algorithm-based code contributions in opposite directions, as predicted. The empirical support for H1, however, is mixed: strong for intrinsic and weak for extrinsic motivational factors.

H2: TABLE 4 and the left side of TABLE 5 (see next two pages) show the effects of motivation (and SWP presence) on reuse-based code contributions (RU2, RU1, LIB). Thus, they report parameter estimates for H2, which contains two predictions about reuse-based contributions: *FOSS developers with above-average monetary and skills-related (extrinsic) motivation are* more *likely to contribute reuse-based code. Developers with above-average joy- and self-expression-related (intrinsic) motivation are* less *likely to contribute reuse-based code.*

Comparing both tables, one can observe the following: monetary motivation increases the odds for above-average reuse-based contributions only in the case of library reuse (+31%). Skill-related motivation increases the odds only in the case of reuse with much adaption (RU2, +35%). Hence, the first prediction of H2 related to extrinsic motivation gets only very weak support. The same holds for the second prediction of H2 related to intrinsic motivation: joy-related motivation decreases the odds for above-average reuse-based contributions only in the case of reuse with little adaption (RU1, -20%). Self-expression-related motivation decreases the odds only in the case of reuse with much adaption (RU2, -22%). The following control variables have an increasing effect on reuse-based contributions: above-average coding time per week (plus 37-55%), patent research (plus 47-65%) and higher education (only for RU1 and LIB, plus 35-47%), and FOSS income (only for RU2 and LIB, plus 38-43%).

TABLE 4: *Effects on reuse-based code contributions (RU2, RU1)*

	Level 2 (RU2)			*Level 1 (RU1)*		
	A	*B*	*C*	*A*	*B*	*C*
SWP LAW	1.282 (0.148)**			0.908 (0.103)		
HI DOM PRESS		0.987 (0.235)			1.380 (0.321)	
LOW DOM PRESS		0.881 (0.144)			1.033 (0.163)	
SWP INCIDENT			1.470 (0.662)			1.262 (0.562)
xM: MONEY	1.175 (0.141)	1.177 (0.141)	1.179 (0.141)	1.221 (0.147)*	1.215 (0.147)	1.222 (0.147)*
xM: SKILLS	1.355 (0.169)**	1.362 (0.170)**	1.350 (0.168)**	1.206 (0.138)	1.208 (0.139)	1.206 (0.138)
iM: JOY	0.822 (0.100)	0.835 (0.101)	0.836 (0.101)	0.801 (0.091)*	0.800 (0.091)**	0.797 (0.091)**
iM: SELF-EXPRESSION	0.782 (0.097)**	0.786 (0.098)*	0.782 (0.097)**	0.969 (0.115)	0.965 (0.115)	0.968 (0.115)
iM: SW FREEDOM	1.359 (0.158)***	1.329 (0.154)**	1.326 (0.154)**	1.284 (0.139)**	1.301 (0.141)**	1.293 (0.140)**
SWP RESEARCH	1.189 (0.207)	1.191 (0.208)	1.182 (0.208)	1.655 (0.282)***	1.602 (0.275)***	1.626 (0.281)***
FOSS INCOME	1.388 (0.192)**	1.403 (0.193)**	1.389 (0.191)**	1.191 (0.162)	1.202 (0.164)	1.184 (0.161)
HIGHER EDU	1.117 (0.165)	1.105 (0.164)	1.108 (0.164)	1.485 (0.212)***	1.474 (0.210)***	1.488 (0.213)***
CODTIME >5	1.556 (0.184)***	1.542 (0.182)***	1.542 (0.182)***	1.372 (0.162)***	1.370 (0.162)***	1.375 (0.162)***
PJEXP >5	1.156 (0.169)	1.139 (0.166)	1.142 (0.166)	0.858 (0.125)	0.854 (0.125)	0.860 (0.125)
AGE >30	1.136 (0.136)	1.176 (0.141)	1.173 (0.140)	1.115 (0.125)	1.112 (0.123)	1.100 (0.122)
aic	2024	2030	2028	2169	2169	2169
bic	2096	2107	2099	2240	2246	2240

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (e^{β}) with robust standard errors. N=1815.

In sum, although the results are significant only in one third of the cases (two out of six coefficient blocks), all significant effects are predicted correctly. Accordingly, the results lend moderate support to H2.

H3: The right side of TABLE 5 shows the effects of motivation (and SWP presence) on reverse-engineering code contributions (REV). Thus, it reports parameter estimates for H3: *FOSS developers with above-average software-freedom-related motivation are more likely to contribute reverse-engineering-based code.*

TABLE 5: *Effects on reuse-based (LIB) and reverse-engineering (REV) code contributions*

	Library-linking (LIB)			*Reverse-engineering (REV)*		
	A	*B*	*C*	*A*	*B*	*C*
SWP LAW	0.788 (0.088)**			1.253 (0.153)*		
HI DOM PRESS		0.952 (0.217)			1.473 (0.370)	
LOW DOM PRESS		1.034 (0.160)			1.310 (0.236)	
SWP INCIDENT			0.750 (0.349)			2.532 (1.156)**
xM: MONEY	1.314 (0.149)**	1.311 (0.149)**	1.308 (0.148)**	1.062 (0.137)	1.058 (0.137)	1.071 (0.138)
xM: SKILLS	1.145 (0.129)	1.144 (0.128)	1.148 (0.129)	1.123 (0.134)	1.109 (0.133)	1.117 (0.134)
iM: JOY	1.186 (0.134)	1.169 (0.132)	1.169 (0.132)	1.141 (0.135)	1.164 (0.137)	1.159 (0.137)
iM: SELF-EXPRESSION	1.057 (0.120)	1.055 (0.120)	1.057 (0.120)	1.021 (0.128)	1.011 (0.127)	1.017 (0.128)
iM: SW FREEDOM	1.099 (0.116)	1.120 (0.118)	1.123 (0.118)	1.060 (0.119)	1.045 (0.117)	1.034 (0.116)
SWP RESEARCH	1.482 (0.255)**	1.468 (0.253)**	1.478 (0.257)**	2.191 (0.369)***	2.210 (0.376)***	2.129 (0.360)***
FOSS INCOME	1.434 (0.194)***	1.420 (0.192)***	1.431 (0.193)***	1.080 (0.157)	1.091 (0.159)	1.071 (0.155)
HIGHER EDU	1.347 (0.182)**	1.358 (0.183)**	1.354 (0.182)**	1.137 (0.170)	1.125 (0.167)	1.132 (0.168)
CODTIME >5	1.429 (0.163)***	1.439 (0.164)***	1.438 (0.164)***	1.341 (0.168)**	1.325 (0.165)**	1.329 (0.165)**
PJEXP >5	0.804 (0.112)	0.816 (0.114)	0.814 (0.114)	0.729 (0.115)**	0.720 (0.114)**	0.718 (0.114)**
AGE >30	1.010 (0.111)	0.977 (0.106)	0.979 (0.106)	0.736 (0.087)***	0.766 (0.090)**	0.755 (0.088)**
aic	2227	2234	2231	2019	2021	2018
bic	2299	2311	2303	2090	2098	2089

* p<0.10, ** p<0.05, *** p<0.01. Estimates displayed as odds ratios (e^{β}) with robust standard errors. N=1815.

None of the motivation factors, including software-freedom, appear to have a significant effect on reverse-engineering-based contributions. The following control variables, however, show significant effects: Patent research (+120%) and above-average time investment (+34%) increase the odds for above-average reverse-engineering-based contributions, while project experience and age decrease the odds (-27%).

Altogether, the results lend no support to H3.

After the first set of motivation-related hypotheses, I now discuss the hypotheses related to SWP presence (H4-H6). The opposing hypotheses on algorithm-based contributions put forward by SWP opponents (H4) and proponents (H5) are discussed first, followed by the hypothesis on reverse-engineering (H6).

H4+H5. While SWP opponents claim that '*stronger SWP presence* reduces *the odds for above-average algorithm-based code contributions by FOSS developers*' (H4), SWP proponents argue for the odds to *increase* (H5).

None of the three SWP-presence variables show to have a consistently significant effect on any of the contribution types. First, the patent law situation increases the odds for above-average reuse-based contribution (RU2, TABLE 4) and decreases them for library reuse (LIB, TABLE 5). Second, neither of the two pressure variables has a significant effect except a high domain pressure that decreases the odds for above-average algorithm-based contributions (AL1, TABLE 3). Third, SWP incidents have no significant effect on either algorithm- or reuse-based contributions. (For the influence of motivational factors and other control variables, see again the discussion for H1.)

In sum, H4 and H5 cannot be confirmed based on the empirical results from this sample as neither hypothesis gets significant support.

H6. The right side of TABLE 5 shows the effects of SWP presence on reverse-engineering-based contributions. Thus, it reports parameter estimates for H6, stating that *stronger SWP presence* increases *the odds for above-average reverse-engineering-based code contributions by FOSS developers*.

TABLE 5 shows that the legal availability of SWP (+25%) and SWP incidents (+153% !) increase the odds for above-average reverse-engineering-based code contributions. Patent pressure does not play a significant role. The control variables show that patent research (+120%) and above-av-

erage coding time (+33%) increase the odds, while above-average project experience (-28%) and being older than 30 years (-25%) decrease the odds for reverse-engineering activities.

Altogether, stronger SWP presence and above-average reverse-engineering contributions appear to go together. Further support is given by the control variable for patent research. The results lend strong support to H6.

6. Conclusion

This study offers a first empirical investigation into the effects of motivation and SWP presence on individual innovation behavior of FOSS developers. A new metric is proposed to measure individual innovation behavior based on code contribution types: in this scale, algorithm-based code contributions are rated more innovative than reuse-based contributions. In a separate analysis, the effect of motivation and SWP presence on reverse-engineering as a special contribution type is analyzed as well. Another new metric is proposed to measure SWP presence: instead of only considering the legal situation of a jurisdiction, the patent pressure within a software domain is also included. A survey was conducted to provide a new data-set for the empirical analysis.

Concerning the effects of motivation on innovation behavior, strong support can be reported for the following result: Above-average intrinsic motivation (joy and self-expression in code-writing) increases the odds for more innovative, algorithm-based code contributions, while above-average extrinsic (monetary and skills-related) motivation seems to decrease the odds. In connection with reuse-based contributions, the opposite relationship finds moderate support: Above-average extrinsic motivation increases the odds for reuse-based contributions, while above-average intrinsic motivation decreases the odds. The third result relates to reverse-engineering: None of the five motivational factors included in the analysis seem to explain why FOSS developers engage in reverse-engineering activities.

These results emphasize the role of motivation within the FOSS system. Particularly intrinsic motivation appears to not only keep this system alive and kicking, but more of it also seems to lead to more innovative contributions. Simply put: ‘Programming challenging new stuff is fun’. On the other side, it appears that reuse-based contributions with a lower innovation level – often needed for ‘the last mile’ before a program is end-user-ready – can be supported by offering extrinsic incentives. What still remains opaque from a theoretical point of view is the question why developers engage in reverse engineering. A broader analysis of motivational factors is needed here.

Concerning the effects of SWP presence on innovation behavior, the empirical results are less conclusive. Neither opponents nor proponents of SWP will find support for their positions that the presence of SWP decrease or increase respectively the odds for innovative, algorithm-based contributions by FOSS developers. None of the three metrics used to capture SWP presence

lends sufficient support to either side – be it positive or negative. Support, however, is found for a hypothesis related to reverse-engineering: stronger SWP presence attracts reverse-engineering-based contributions by FOSS developers.

These results confirm several challenges for research as well as for policy-makers. Both continue to lack a broad, sound empirical foundation to discuss the effects of software patents on FOSS innovation.

For researchers, the challenges raised in this study are (a) to develop an easy-to use yet non-trivial metric to measure the presence of software patents empirically; (b) to quantify their effect on the FOSS system, helping policy-makers make better-informed decision. For future research, it would be useful to verify some of the links argued for in this study using other data sources. CVS logs have been used in the past for code contribution analysis. Maybe the innovation metric proposed here could be helpful in that regard.

For policy-makers in innovation and intellectual property policy fields the challenges are (a) to decide whether FOSS deserves a special case when debating software patents because of its unique way of producing software for the common good; (b) to continue treading carefully in the field of software patents before jumping to legislation. The FOSS market has reached a size where harm cannot be considered collateral damage as it may have in the past. Although the results have not shown systematic harm to the FOSS communities, there is still no empirical support that the traditional arguments in favor of patents do hold for the FOSS system – or software in general as some continue to argue.

Some limitations of the study deserve mentioning. First, taking the individual developer as unit of analysis ignores explanatory factors on project level that can also influence innovation behavior, such as project size and organizational structure. The larger a project is, the more elaborate its organization structure becomes, the more contributors tend to specialize in their contributions – up to a point where dedicated roles may emerge. Such a division of labor biases the measurement of individual innovation behavior. Second, it is impossible to investigate whether software patents caused projects to stop by only surveying ‘alive’ projects from SF as it has been done in this study. To obtain a complete picture, it is necessary to run a dedicated study on failed projects – even if the response rate will be very low.

7. References

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

Allison, P.D. (2002). *Missing Data*. Thousand Oaks, CA: Sage Publications.

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

Bessen, J., Hunt, R. (2007). An Empirical Look at Software Patents. *Journal of Economics & Management Strategy*, (16)1, 157-189.

Bessen, J., Hunt, R. (2004). The Software Patent Experiment. In OECD (Ed.), *Patents, Innovation and Economic Performance* (pp. 247-263). Paris: OECD Publishing.

Bessen, J., Maskin, E. (2006). *Sequential Innovation, Patents, and Imitation*. Institute for Advanced Study, School of Social Science, Working Paper Economics No. 0025. http://www.researchoninnovation.org/patrev.pdf

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

Boldrin, M., Levine, D.K. (2008). *Against intellectual monopoly*. New York: Cambridge University 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.

Czaja, R., Blair, J. (1996). *Designing surveys : a guide to decisions and procedures*. Thousand Oaks, CA: Pine Forge Press.

Dapp, M. (2009). *Nothing Really Matters? Empirical Evidence on The Effects of Software Patents On the Motivation of Free/Open Source Software Developers*. Available at SSRN: http://ssrn.com/abstract=1422720 (19.06.09).

Dapp, M., Bernauer, T. (2009). *Hot Debate about Chilling Effects : Do Software Patents Hamper Free/Open Source Software Development?*. Center for Comparative and International Studies, ETH Zurich, Working Paper 40/2009.

Dillman, D.A. (2000). *Mail and internet surveys: the tailored design method*. New York: Wiley.

English, R., Schweik, C.M. (2007). Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. *UPGRADE*, (IX)6, 54-59.

European Commission (2002). *DIRECTIVE OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the patentability of computer-implemented inventions COM(2002) 92 final*. Retrieved from http://eur-lex.europa.eu/LexUriServ/site/en/com/2002/com2002_0092en01.pdf.

European Information and Communications Technology Industry Association (EICTA) (2000). *Response to the European Commission's Consultation Paper on "The Patentability of computer-iomplemented inventions"*. Retrieved 05.02.2009 from ec.europa.eu/internal_market/indprop/docs/comp/replies/eicta_en.pdf.

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.

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

Foundation for a Free Information Infrastructure (2009). *Software Patents in Action*. Retrieved 13.03.2009 from http://eupat.ffii.org/patents/effects/.

Fowler, F.J. (2002). *Survey Research Methods*. Thousand Oaks, CA: Sage Publications.

Fowler, F.J. (1995). *Improving Survey Questions : Design and Evaluation*. Thousand Oaks, CA: Sage Publications.

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

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.

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

Howison, J., Conklin, M., Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, (1)3, 17-26.

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

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

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.

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.

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.

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.

OECD (1997). *Oslo Manual : Proposed guidelines for collecting and interpreting technological innovation data*. : OECD.

Rogers, E. (2003). *Diffusion of Innovations*. New York: Free Press.

Royston, P. (2005). Multiple imputation of missing values: update. *The Stata Journal*, (5)2, 188-201.

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.

Samuelson, P., Scotchmer, S. (2002). The Law and Economics of Reverse Engineering. *The Yale Law Journal*, (111), 1575-1663.

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

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

Smith, K. (2005). Measuring Innovation. In Fagerberg, J., Mowery, D.C., Nelson, R.R. (Eds.), *The Oxford Handbook of Innovation* (pp. 148-177). Oxford: Oxford University Press.

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

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.

Watts, D.J. (2003). *Six degrees: The science of a connected age*. New York: Norton.

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

8. Appendix A – Descriptive statistics

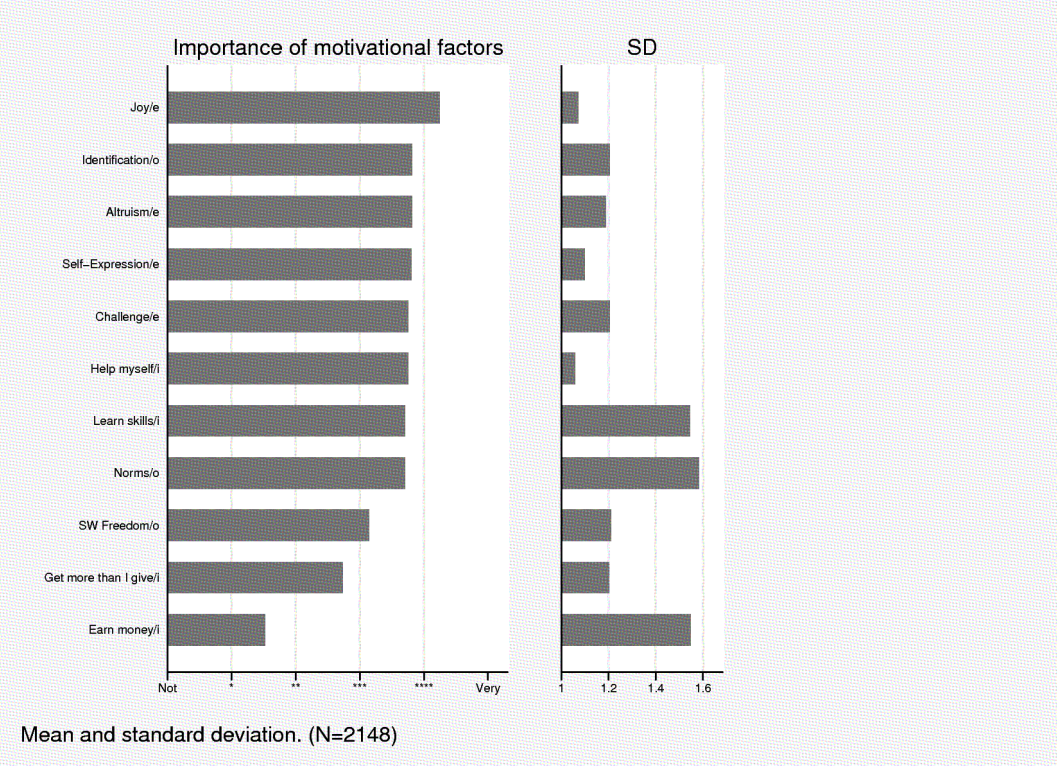

FIGURE-A 1: *Reported importance of motivational factors: (e)njoyment, (o)bligation, and (i)nstant*

9. Appendix B – Underlying survey questions

Here, the questions and answer categories from the survey questionnaire used for this study are listed. The variable names used in the regression analysis are mentioned in bracketed capitals.

Code contributions. When you contributed code to <project-name> during the last two years, how did you typically create source code?

	0 Never	X	XX	XXX	XXXX	XXXX X Always	No answer
By linking to existing FLOSS libraries. (LIB)	O	O	O	O	O	O	O
By integrating existing FLOSS components with little adaption. (REUSE 1)	O	O	O	O	O	O	O
By recombining existing FLOSS components with much adaption. (REUSE 2)	O	O	O	O	O	O	O
By reverse-engineering/imitating functionality from non-FLOSS programs. (REV)	O	O	O	O	O	O	O
By coding known algorithms/methods from scratch. (ALGORITHM 1)	O	O	O	O	O	O	O
By inventing new algorithms/methods before coding. (ALGORTIHM 2)	O	O	O	O	O	O	O

Motivation. Please rank how important the following motives are for you to contribute code to FLOSS projects!

	- - - Not at all	- -	-	+	+ +	+ + + Very	No answer
I simply enjoy programming. (JOY)	O	O	O	O	O	O	O
I want to create beautiful and elegant programs. (SELF-EXPRESSION)	O	O	O	O	O	O	O
I seek the challenge of solving programming tasks.	O	O	O	O	O	O	O
I feel good about helping others with my programs.	O	O	O	O	O	O	O
I value the goals of the FLOSS community.	O	O	O	O	O	O	O
I support the technical goals of my project.	O	O	O	O	O	O	O
I think software should be free. (SOFTWARE FREEDOM)	O	O	O	O	O	O	O
How my family/friends see my engagement is ...	O	O	O	O	O	O	O
I get back more than I contribute.	O	O	O	O	O	O	O
I develop my programming skills. (SKILLS)	O	O	O	O	O	O	O
I solve my own programming problems.	O	O	O	O	O	O	O
I earn money from it. (MONEY)	O	O	O	O	O	O	O
I build a reputation as a good developer.	O	O	O	O	O	O	O

	- - - Not at all	- -	-	+	+ +	+ + + Very	No answer
I improve my future career perspectives.	O	O	O	O	O	O	O
I think it is fair to give back when you take.	O	O	O	O	O	O	O
I fight against proprietary software.	O	O	O	O	O	O	O

Software patent influence. According to your knowledge, which application domains are mostly affected by software patents? Select a maximum of four (4) domains!

O	Communications	O	Database	O	Desktop Environment
O	Education	O	Formats and Protocols	O	Games/Entertainment
O	Internet	O	Multimedia	O	Office/Business
O	Scientific/Engineering	O	Security	O	Software Development
O	System	O	Text Editors		

Main Appendix A – Survey Questionnaire

Please note:

- The online survey questionnaire was implemented using the survey tool LimeSurvey.org, which uses typical interactive web features like drop-down boxes, etc. These features cannot be replicated directly in this document. Instead, the options respondents were able to choose from are shown as a list.

- Also, the survey tool offered to show a different question flow depending on the answers to previous questions. Such forking is shown with directional statements in brackets such as "[Only answer this question if ...]". Aside from that, the questions, answers, and comments are shown as in the original survey questionnaire.

- The questionnaire starts with a short introduction page helping the respondents using the online tool. Questions are presented in six sections A to F. Depending on the answers given, some sections were skipped and not visible for the respondent.

- Finally, the text also contains automatic text fields (e.g., <project-name>) that were used to personalize the questionnaire.

FLOSS, Software Patents, and Innovation

Dear <SF-login-name>, please read the following carefully before you start!

What? Background information about the research project behind this survey can be found here. The project is financed through ETH research grant No. TH -2/05-2.

SSL In case you did not get an SSL connection automatically, please add an 's' after 'http' in the URL before you continue. The SSL certificate belongs to ETH Zurich, Switzerland.

Javascript must be enabled to allow the survey to function properly. You can browse back and forth as long as you have not pressed the [submit] button on the last page. Please note that questions marked as mandatory (*) need an answer.

Answers are not saved automatically page by page! Only when you press [submit] at the end of the survey, all answers are saved at once. If you allow cookies, you can interrupt the survey, save your answers (password protected) and continue later.

What do we do to protect your **privacy** knowing that this is a sensitive topic? In general, we adhere to the Guide on Ethical Online Research by the Association of Internet Researchers (AOIR). In particular, SF login, email address and projectname are only used to communicate with you during this survey. They will be removed from the final data-set and at no time be passed on to third parties. Only aggregated results (not on project level) will be published, so no conclusions can be drawn from the results to identify projects.

Results will be made available on this website in an open data format and licensed under a suitable permissive copyright license (e.g., CC). We thank all free/libre/open source software (FLOSS) communities for providing the software to make this survey possible, particularly LimeSurvey.org and FLOSSmole.

Thanks for your trust and your invaluable contribution! And good luck with the lottery. :-)

Best regards
Prof. Thomas Bernauer, ETH Zurich CIS
Prof. Georg von Krogh, ETH Zurich SMI
Prof. Gérard Hertig, ETH Zurich L&E
Marcus M. Dapp, ETH Zurich CIS/SMI

Part A - Your experience

a1-comm: Community. In this survey, we will always use the umbrella term 'Free/Libre Open Source Software (FLOSS)', knowing that many developers make a distinction. Do you think of yourself as part of the Free/Libre Software or of the Open Source community?

Please choose only one of the following:

I think of myself as part of the Free/Libre Software community.

I think of myself as part of the Open Source Software community.

I do not make this distinction.

a2-exp: Years of experience. What year did you start contributing source code to FLOSS projects?

Please choose only one of the following:

2006

2005

2004

2003

2002

2001

2000

(...)

1989

1988

1987

1986

1985

1984

Earlier than 1984

a3-pjs: Project track record. To how many FLOSS projects have you been actively contributing source code since then?

Please choose only one of the following:

1

2-5

6-10

11-20

21-30

30+

a4-insp: Sources of inspiration. Where from do you get ideas/input/inspiration for code contributions?

Please select a maximum of three (3) sources.

Please choose all that apply:

From my own needs/use.

From other FLOSS projects.

From other proprietary programs.

From feature requests.

From bug reports.

From talks with people inside our project.

From talks with people outside the project.

From research in books, magazines, etc.

From web research in blogs, wikis, fora, etc.

From reading published patents.

From 'flashes of genius' (e.g., in the shower/bathtub).

From my daily work.

Other: _____

* a5-motive: Motivation. Please rank how important the following motives are for you to contribute code to FLOSS projects!

Please choose the appropriate response for each item:

	Not at all important ---	--	-	+	++	Very important +++	No answer
I simply enjoy programming.	O	O	O	O	O	O	O
I want to create beautiful and elegant programs.	O	O	O	O	O	O	O
I seek the challenge of solving programming tasks.	O	O	O	O	O	O	O
I feel good about helping others with my programs.	O	O	O	O	O	O	O
I value the goals of the FLOSS community.	O	O	O	O	O	O	O
I support the technical goals of my project.	O	O	O	O	O	O	O
I think software should be free.	O	O	O	O	O	O	O
How my family/friends see my engagement is ...	O	O	O	O	O	O	O
I get back more than I contribute.	O	O	O	O	O	O	O
I develop my programming skills.	O	O	O	O	O	O	O
I solve my own programming problems.	O	O	O	O	O	O	O
I earn money from it.	O	O	O	O	O	O	O
I build a reputation as a good developer.	O	O	O	O	O	O	O
I improve my future career perspectives.	O	O	O	O	O	O	O
I think it is fair to give back when you take.	O	O	O	O	O	O	O
I fight against proprietary software.	O	O	O	O	O	O	O

* a6-income: Income. Was developing FLOSS your main source of income in the last two years, i.e. the period August 2005 until August 2007?

The "last two years" in this survey always means the period from August 2005 until August 2007.

Please choose only one of the following:

I was paid mainly for developing FLOSS.

I was paid for FLOSS-related work (e.g., admin, support) with no or only little FLOSS code development work.

I was not paid for any FLOSS-related work.

* a7-time: Time. On average, how many hours per week - work and free time combined - did you contribute source code to FLOSS projects in the last two years? How many hours per week have you been paid on average?

	0h	<2h	2-5h	6-10h	11-20h	21-40h	>40h	No answer
In an average week, I coded for ...	O	O	O	O	O	O	O	O
From that time, I have been paid for ...	O	O	O	O	O	O	O	O

* a8-icsrc: Sources of code. When you contributed code to <project-name> during the last two years, how did you typically create source code?

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
By linking to existing FLOSS libraries.	O	O	O	O	O	O	O
By integrating existing FLOSS components with little adaption.	O	O	O	O	O	O	O
By recombining existing FLOSS components with much adaption.	O	O	O	O	O	O	O
By reverse-engineering/imitating functionality from non-FLOSS programs.	O	O	O	O	O	O	O
By coding known algorithms/methods from scratch.	O	O	O	O	O	O	O
By inventing new algorithms/methods before coding.	O	O	O	O	O	O	O

Part B - The project

* b1-appdom: Application domain. Under which primary category in SourceForge's Software Map is your project registered?

If your project is listed under more than one category, please choose the primary/main category. If your primary category is any of {terminals|printing|sociology|religion} and is not listed here, please choose "Other".

Please choose only one of the following:

Communications

Database

Desktop Environment

Education

Formats and Protocols

Games/Entertainment

Internet

Multimedia

Office/Business

Scientific/Engineering

Security

Software Development

System

Text Editors

Other category or not categorized.

* b2-codesrc: Sources of code. When you think about all code contributions to <project-name> during the last two years, how did the developers typically create source code?

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
By linking to existing FLOSS libraries.	O	O	O	O	O	O	O
By integrating existing FLOSS components with little adaption.	O	O	O	O	O	O	O
By recombining existing FLOSS components with much adaption.	O	O	O	O	O	O	O
By reverse-engineering/imitating functionality from non-FLOSS programs.	O	O	O	O	O	O	O
By coding known algorithms/methods from scratch.	O	O	O	O	O	O	O
By inventing new algorithms/methods before coding.	O	O	O	O	O	O	O

* b3-position: Positioning. How would you rate the <project-name> project compared to similar, competing projects during the last two years?

Please choose the appropriate response for each item:

	Below average (-)	Average (=)	Above average (+)	No answer
In terms of innovative new functionality, we are	O	O	O	O
In terms of growing the user base, we are	O	O	O	O
In terms of vibrancy of the developer community, we are	O	O	O	O

Part C - Software patents in general

c1-patknow: Patent knowledge. My knowledge about ...

Please choose the appropriate response for each item:

	Zero o	Poor x	Fair x x	Good x x x	Excellent x x x x	No answer
how patents work in general is ...	O	O	O	O	O	O
the software patent debate in general is ...	O	O	O	O	O	O
the software patent situation in my country is ...	O	O	O	O	O	O

c2-affdoms: Software patent influence. According to your knowledge, which application domains are mostly affected by software patents?

Select a maximum of four (4) domains!

Please choose all that apply:

- Communications
- Database
- Desktop Environment
- Education
- Formats and Protocols
- Games/Entertainment
- Internet
- Multimedia
- Office/Business
- Scientific/Engineering
- Security
- Software Development
- System
- Text Editors

* c3-natlaw: National law. According to your knowledge, did the national law of your country allow software patents during the last two years?

This question is about the situation according to the law, not the practice of the patent office.

Please choose the appropriate response for each item:

	Yes	No	No answer
The law allowed software patents:	O	O	O

* c4-patoff: Patent offices. According to your knowledge, how frequently were software patents granted in the last two years...

This question is about the patent office practice, not the legal situation (law).

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
by the national patent office in -your- country?	O	O	O	O	O	O	O
by the European Patent Office for European countries?	O	O	O	O	O	O	O

Part D - The project & software patents

* d1-patres: Patent Research. Have developers of the <project-name> project ever done research in patent databases?

Research can be done for different reasons: to check for infringements, to get new coding ideas, etc.

Please choose the appropriate response for each item:

	Yes	No	No answer
Patent research was done:	O	O	O

* d2-patrole: Role of patents. From what you know, have software patents actually played any relevant role for the <project-name> project?

Please focus on facts and your experience in the project and avoid opinions you may have otherwise.

Please choose only one of the following:

Yes, patents played a negative role.

Yes, patents played a positive role.

No, patents did not play any role.

No answer.

Part D2 - The project & software patents (cont'd.)

[Only answer this question if you answered 'Yes, patents played a negative role.' to question 'd2-patrole ']

dn1: Incidents. Was the project ever contacted by someone raising patent claims on functionality in <project-name>s code base?

Please choose only one of the following:

No, never.

Yes, once.

Yes, several times from the same patent holder.

Yes, several times from different patent holders.

No answer.

[Only answer this question if you answered 'Yes, once.' or 'Yes, several times from the same patent holder.' or 'Yes, several times from different patent holders.' to question 'dn1 ' and if you answered 'Yes, patents played a negative role.' to question 'd2-patrole ']

dn2: Patent holder. Who was raising patent claims?

Information and Communication Technologies (ICT) include more than just software.

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
A software company offering a competing program.	O	O	O	O	O	O	O
A software company that is not directly competing.	O	O	O	O	O	O	O
An ICT company that is not primarily software-producing.	O	O	O	O	O	O	O
A company outside the ICT field.	O	O	O	O	O	O	O
A company acquiring patents for the purpose of asserting them against others ("patent troll").	O	O	O	O	O	O	O
Other.	O	O	O	O	O	O	O

[Only answer this question if you answered 'Yes, once.' or 'Yes, several times from the same patent holder.' or 'Yes, several times from different patent holders.' to question 'dn1 ' and if you answered 'Yes, patents played a negative role.' to question 'd2-patrole ']

dn3: Incident evolution. How did this/these incident/s evolve?

"Letters" can be on paper or via email.

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
We received a written letter of notice offering a license for free.	O	O	O	O	O	O	O
We received a written letter of notice offering a license against a fee.	O	O	O	O	O	O	O
We received a formal cease-and-desist letter threatening to take legal steps.	O	O	O	O	O	O	O
We were sued.	O	O	O	O	O	O	O

[Only answer this question if you answered 'No, never.' or 'No answer.' to question 'dn1 ' and if you answered 'Yes, patents played a negative role.' or 'Yes, patents played a positive role.' to question 'd2-patrole ']

dn3-alt: Clarification. Please clarify briefly: What actually happened in the project related to software patents?

Thanks for providing a few sentences of clarification.

Please write your answer here:

[Only answer this question if you answered 'Yes, patents played a positive role.' to question 'd2-patrole ']

dn3-pos: Positive role. Please describe briefly what positive role software patents played in your project!

Please write your answer here:

[Only answer this question if you answered 'Yes, once.' or 'Yes, several times from the same patent holder.' or 'Yes, several times from different patent holders.' to question 'dn1 ' and if you answered 'Yes, patents played a negative role.' to question 'd2-patrole ']

dn4: Short-term. How did the <project-name> project react?

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
We ignored the request and continued development.	O	O	O	O	O	O	O
We complied by making the code non-infringing.	O	O	O	O	O	O	O
We complied by signing a license deal.	O	O	O	O	O	O	O
We defended the project legally.	O	O	O	O	O	O	O

[Only answer this question if you answered 'Yes, once.' or 'Yes, several times from the same patent holder.' or 'Yes, several times from different patent holders.' to question 'dn1 ' and if you answered 'Yes, patents played a negative role.' to question 'd2-patrole ']

dn5: Long-term. What were the consequences for the <project-name> project in the long run?

Please choose the appropriate response for each item:

	Never o	x	x x	x x x	x x x x	Always x x x x x	No answer
We had delays in code development.	O	O	O	O	O	O	O
We had to use a non-infringing, inferior substitute.	O	O	O	O	O	O	O
We came up with a non-infringing, superior substitute.	O	O	O	O	O	O	O
We had to leave out infringing functionality with no substitute.	O	O	O	O	O	O	O
We had to pay license fees to comply.	O	O	O	O	O	O	O
We had legal defense costs.	O	O	O	O	O	O	O
People contributed less code.	O	O	O	O	O	O	O
People left the project.	O	O	O	O	O	O	O
The project's existence was in danger.	O	O	O	O	O	O	O
Other consequences.	O	O	O	O	O	O	O

[Only answer this question if you answered 'Yes, patents played a negative role.' or 'Yes, patents played a positive role.' to question 'd2-patrole ']

dn6: Effects on motivation. How would you rate the effects of all that on developers' motivation?

Please choose the appropriate response for each item:

	Decreased (-)	Stayed the same (=)	Increased (+)	No answer
My personal motivation:	O	O	O	O
Overall motivation of the team:	O	O	O	O

Part E - Project Environment

* e1: Licensing. According to SourceForge data from August 2006, the source code of <project-name> is licensed under the following license(s): <project-license>.

Please choose the appropriate response for each item:

	Yes	No	No answer
Did software patents play a role in choosing the license(s) in the past?	O	O	O
Are there plans to switch to other licenses containing software patent provisions (e.g. GPLv3)?	O	O	O

e2: Protection from patents. Different organisations offer legal protection schemes for FLOSS projects through patent pools, pledges or insurances. What is the situation with <project-name> in that regard?

Examples for such organisations are Open Invention Network (OIN), Patent Commons or OSRM.

Please choose all that apply:

We seek to become a member of a protection scheme.

We are member of a protection scheme.

We seek to get a patent litigation insurance.

We have a patent litigation insurance.

* e3: Legal representation. Does an organisation represent the <project-name> project in legal matters?

Please choose only one of the following:

Yes, a non-profit foundation/association.

Yes, a for-profit company.

Yes, an academic institution/university.

No, we are just a group of individuals.

Don't know.

[Only answer this question if you answered 'Yes, a non-profit foundation/association.' or 'Yes, a for-profit company.' or 'Yes, an academic institution/university.' to question 'e3 ']

e5: Country. In which country was that organisation located in the last two years?

Please choose only one of the following:

Argentina

Australia

Austria

Belgium

Brazil

Canada

Chile

China

Cyprus

Czech Republic

Denmark

Estonia

Finland

France

Germany

Greece

Hungary

Iceland

India

Ireland

Israel

Italy

Japan

Latvia

Lithuania

Luxembourg

Malta

Mexico

Netherlands

New Zealand

Norway

Poland

Portugal

Romania

Russian Federation

Singapore

Slovakia

Slovenia

South Africa

South Korea

Spain

Sweden

Switzerland

Turkey

United Kingdom

United States of America

Other

[Only answer this question if you answered 'Yes, a for-profit company.' or 'Yes, a non-profit foundation/association.' or 'Yes, an academic institution/university.' to question 'e3 ']

e6: Rights. Does that organisation hold legal rights to the project's code base?

Please choose the appropriate response for each item:

	Yes	No	No answer
It holds parts of the copyrights to the code base.	O	O	O
It holds all copyrights to the code base.	O	O	O
It holds software patents related to the code base.	O	O	O

[Only answer this question if you answered 'No' to question 'e6 ' and if you answered 'Yes, a non-profit foundation/association.' or 'Yes, a for-profit company.' or 'Yes, an academic institution/university.' to question 'e3 ']

e7ra: Reasons against software patents. Why does the organisation not hold own software patents?

The reasons are shown in random order.

Please choose all that apply:

Software patents give too little protection.

Infringements are hard to prove.

The granting process for software patents is long.

To apply for and enforce software patents is expensive.

The code's innovativeness is too low for software patents.

Doubts to disclose know-how contained in the code.

Software patents are not relevant for its business model.

Knowledge in the patent application process is lacking.

Doubts whether software patents hinder innovation in software.

Doubts about the image of software patents in the market.

Doubts about the patentability of software in general.

Doubts about the enforceability of software patents.

Other.

[Only answer this question if you answered 'Yes' to question 'e6 ' and if you answered 'Yes, a non-profit foundation/association.' or 'Yes, a for-profit company.' or 'Yes, an academic institution/university.' to question 'e3 ']

e7rf: Reasons for software patents. Why does the organisation hold own software patents?

The reasons are shown in random order.

Please choose all that apply:

- To protect itself from imitation.
- To Increase its market advantage.
- To block out competition.
- Patents are important in the EU.
- Patents are important in the US.
- Patents are important in Japan.
- To increase the software's financial value.
- To get better access to the capital market.
- To have a portfolio for cross-licensing deals.
- To get access to patent pools.
- To generate income from licensing.
- Other.

[Only answer this question if you answered 'Yes' or 'No' to question 'e6 ' and if you answered 'Yes, a non-profit foundation/association.' or 'Yes, a for-profit company.' or 'Yes, an academic institution/university.' to question 'e3 ']

e8: Comment about reasons. If you want to make a comment about the reasons, please do so here!

Please write your answer here:

e9: Company involvement. Are companies directly involved in the software development for the project?

An example of direct involvement is when the company has hired developers to write code for the project.

Please choose only one of the following:

Yes, more than one company.

Yes, one company.

No.

Part F - Demographics

* f1-resid: Country. What was your main country of residence in the last two years? If you have been living in different countries, please choose the one you have been living in longest.

Please choose "other" (at the end of the list) if your country is not listed.

Please choose only one of the following:

Argentina

Australia

Austria

Belgium

(... abbreviated, see question ‘e5’ for full country list ...)

Switzerland

Turkey

United Kingdom

United States of America

Other.

f2-nation: Nationality. Which country's nationality do you have?

Please choose "other" (at the end of the list) if your country is not listed.

Please choose only one of the following:

Argentina

Australia

Austria

Belgium

(... abbreviated, see question ‘e5’ for full country list ...)

Switzerland

Turkey

United Kingdom

United States of America

Other.

f3-employ: Employment. How would you describe your main employment situation in the last two years?

Please choose only one of the following:

Self-employed.

Employed.

Unemployed.

Student, unpaid work.

Other.

f4-edu: Level of education. What is the highest level of eduction that you completed?

Please choose only one of the following:

Elementary School

High School

Apprenticeship

Bachelor (university)

Master (university)

Ph.D. (university)

f5-sex: Gender. What is your gender?

Please choose only one of the following:

Female

Male

f6-age: Age group. To which age group do you belong?

Please choose only one of the following:

15 yrs and less

16-20 yrs

21-25 yrs

26-30 yrs

31-35 yrs

36-40 yrs

41-45 yrs

46-50 yrs

51-55 yrs

56-60 yrs

61 yrs and more

No answer

f7-comment: Final comment. Is there anything else you want to tell us?

Please write your answer here:

Thank you for completing this survey!

Main Appendix B – Invitation Email

Dear {FIRSTNAME}!

There is considerable debating in the FLOSS community about software patents; but what do we really know about their effects? What are your own experiences? To get a representative picture on the software patent situation, we cordially invite you to participate in this global scientific survey:

{SURVEYNAME} -- {SURVEYDESCRIPTION}

»» {SURVEYURL}

Participation is customized and by invitation only!

You have been randomly selected from project leaders/key developers of all SF projects active in August 2006. To improve statistical power of the survey, we are using a controlled survey setting with direct email invitations. You can rest assured that all data will be aggregated and completely anonymous. Details on data handling can be found following the survey link.

The survey is a joint project of the Center for Comparative and International Studies (CIS) and the Chair for Strategic Management and Innovation (SMI) at ETH Zurich, Switzerland[1]. Financial support through ETH Research Grant No. TH-2/05-2 is kindly acknowledged.

To show we are serious to have you participate, we invite all participants who complete the questionnaire to participate in our lottery. The cool prizes we raised for this dedicated group of individuals are:

1st prize - A "One Laptop Per Child" device, sponsored by Google's Open Source Program Office[2]

2nd prize - A Neo1973 free-your-phone, sponsored by OpenMoko/FIC[3]

3rd prize - In progress. We aim for similar coolness as the other prizes :-)

Please submit your response before: ..., xx.yy.2007!

Answering the multiple choice questions will be easy. We hope you find coming up with answers as exciting as we found coming up with questions. Thank you very much for your interest, time and invaluable contribution!

With kind regards,

Professor Thomas Bernauer, CIS
Professor Georg von Krogh, SMI
Professor Gérard Hertig, L&E
Marcus M. Dapp, CIS/SMI

If you face technical problems, email Marcus at {ADMINEMAIL}

[1] http://www.ethz.ch, http://www.smi.ethz.ch, and http://www.cis.ethz.ch

[2] http://www.laptop.org

[3] http://www.openmoko.org

Marcus M. Dapp | WEC C 19 | ETH-Zentrum | CH-8092 Zurich | Switzerland

Main Appendix C – First Reminder Email

Dear {FIRSTNAME},

We want to remind you that our research study, which is surveying the potential effects of software patents on free/libre/open source projects, is still open. -- This is a joint research project [0] of the Center for Comparative and International Studies (CIS), the Chair for Strategic Management and Innovation (SMI), and the Chair for Law&Economics; all at the Federal Institute of Technology (ETH) in Zurich, Switzerland: http://www.ethz.ch. It is fully funded by an ETH research grant.

We make sure that all participating projects/individuals remain anonymous and all results will be aggregated as we are aware of the sensitive nature of the research topic. Identifying information (login/projectname/email address) is only used for this email conversation and only handled by one individual. More about our privacy policy is on the information page before the actual questionnaire.

Participation is by invitation only. We are including only a random sample of project leaders/key developers of SF projects as of August 2006. Therefore, your contribution is crucial and will directly improve the overall data quality.

Thanks for helping us by submitting your response within the next days. Multiple choice questions will make answering straightforward. If you face technical problems, please email Marcus at {ADMINEMAIL} with the subject line: 'bug-report-ig'.

=> {SURVEYURL}

As an additional benefit besides publishing the results, we include all completed questionnaires in a lottery.

1st -- A green 'XO' (OLPC) laptop, sponsored by Google's Open Source Program Office[1]

2nd -- A free 'Neo1973' mobile phone, sponsored by OpenMoko/FIC[2]

3rd -- Be surprised. We aim for similar 'coolness' as the other prizes ;-)

Thank you very much for your time.

Professor Thomas Bernauer, http://www.cis.ethz.ch
Professor Georg von Krogh, http://www.smi.ethz.ch
Professor Gérard Hertig, http://www.hertig.ethz.ch
Marcus M. Dapp, Ph.D. candidate & survey responsible

[0] https://www.rdb.ethz.ch/projects/project.php?proj_id=13158

[1] http://www.laptop.org

[2] http://www.openmoko.org

Marcus M. Dapp | WEC C 19 | ETH-Zentrum | CH-8092 Zurich | Switzerland

Main Appendix D – Second (Last) Reminder Email

Dear {FIRSTNAME},

The research study of ETH Zurich on software patents and FLOSS projects is still open. If you want to discard this email, please read it before you do so. :-)

1) This study adheres to academic standards. This is a joint research project [0] of the Center for Comparative and International Studies (CIS), the Chair for Strategic Management and Innovation (SMI), and the Chair for Law&Economics; all at the Federal Institute of Technology (ETH) in Zurich, Switzerland: http://www.ethz.ch. It is fully funded by an academic research grant by ETH. We added an SSL certificate to the survey website to prove who we are.

2) This is a sensitive, legal issue for some projects. We know. Therefore, we do all we can to ensure that all participating projects/individuals remain anonymous and all results will be aggregated. Identifying information (login/projectname/email address) is only used for this email conversation and only handled by one individual. More details about our privacy policy is on the information page before the actual questionnaire.

3) Why is participation by invitation only? Because we want a random sample of projects, not just the ones with/or without software patent issues. In fact, we do not know anything about your project in connection with software patents; it was randomly drawn. Getting a representative picture about the software patent situation is the goal of the survey. Therefore, -all- answers are important. In other words, please participate, no matter what the situation in your project is. You can explain us in the survey.

4) Interesting, but I am not the right person. We are including only a random sample of project leaders/key developers of SF projects as of August 2006. Therefore, your contribution is crucial and will directly improve the overall data quality. If, for whatever reason, you think someone else is better suited to represent the {LASTNAME} project from a project leader/key developer perspective in the period Aug05-Aug07, we would appreciate if you could send us an alternative email address to invite that person. (Yours will be removed.)

Thanks for helping this academic endeavor by submitting your response within the next days. If you face technical problems, please email Marcus at {ADMINEMAIL} with the subject line: 'bug-report-sg'.

=> {SURVEYURL}

As an additional benefit to you, besides orderly publishing of the results, we include all completed questionnaires in a lottery with prizes:

1st: A green 'XO' (OLPC) laptop, sponsored by Google's Open Source Program Office[1]

2nd: A free 'Neo1973' mobile phone, sponsored by OpenMoko/FIC[2]

3rd: Working on it; will be comparable to the 1st prize in nature.

Thank you very much for your time.

Professor Thomas Bernauer, http://www.cis.ethz.ch
Professor Georg von Krogh, http://www.smi.ethz.ch
Professor Gérard Hertig, http://www.hertig.ethz.ch
Marcus M. Dapp, Ph.D. candidate & survey responsible

[0] https://www.rdb.ethz.ch/projects/project.php?proj_id=13158

[1] http://www.laptop.org

[2] http://www.openmoko.org

Marcus M. Dapp | WEC C 19 | ETH-Zentrum | CH-8092 Zurich | Switzerland

Main Appendix E – Imputation statistics

The following table lists all variables and the fraction of missing and hence imputed values.

Variable name	# Missing	% Missing
Experience [years]	6	0,2
Experience [projects]	11	0,5
Motivation: joy	24	1
Motivation: beauty	32	1,3
Motivation: challenge	29	1,2
Motivation: helping others	21	0,9
Motivation: community goals	51	2,1
Motivation: project goals	97	4
Motivation: SW must be free	68	2,8
Motivation: significant others	170	7
Motivation: net gain	77	3,2
Motivation: skills	35	1,4
Motivation: self-help	44	1,8
Motivation: money	65	2,7
Motivation: reputation	39	1,6
Motivation: future career	42	1,7
Motivation: reciprocity	51	2,1
Motivation: fight proprietary SW	76	3,1
Income	0	0
Coding time: total	61	2,5
Coding time: paid	159	6,5
Code contribution: LIB	181	7,4
Code contribution: RU1	183	7,5
Code contribution: RU2	191	7,8
Code contribution: AL1	157	6,4
Code contribution: AL2	174	7,1
Code contribution: REV	181	7,4
Application domain of project	0	0
Patent research	256	10,5

Variable name	# Missing	% Missing
Role of SWP	212	8,7
Patent claims against project	222	9,1
Country of residence	0	0
Nationality	25	1
Employment	12	0,5
Education	22	0,9
Sex	29	1,2
Age group	30	1,2