Some Economic & Legal Aspects of Open Source Software

Jonathon J. Frost[1]
University of Washington
Department of Economics

Mentors[2]
Prof. Keith Leffler
University of Washington
Department of Economics

Prof. Robert Gomulkiewicz
University of Washington
School of Law

Prof. Dan Laster
University of Washington
School of Law

This version: May 7th, 2005

Abstract
The emergence of open source software as a viable economic model has risen to the forefront in the debate on the future of the information technology industry. However, at first glance, the open source software development model is strikingly enigmatic and counterintuitive. To help better understand this phenomenon, this paper, through market data and economic theory, proceeds to ask and answer three related questions. First, what is the economic relationship between open source software development communities and proprietary software firms? Second, what are the resulting effects on market innovation and innovation incentives? And third, what legal mechanisms allow for the sustainability of open source software and should they be expanded or reduced? This paper concludes that open source activity appears to be generating four economic effects, whose net affect on innovation in the software market is ambiguous.

---

[1] Jonathon J. Frost is a senior at the University of Washington majoring in Economics (Honors Program) and can be contacted at jjfrost@u.washington.edu. This paper as well as its related topic proposal and other relevant materials can be found at the following website: http://students.washington.edu/jjfrost/.

[2] I would like to express my gratitude to my mentors, Prof. Keith Leffler, Prof. Bob Gomulkiewicz and Prof. Dan Laster for taking the time out of their busy schedules to give me such helpful advice and guidance. I would also like to thank Prof. Judith Thornton for her thoughtful suggestions during the early stages of this paper (and later ones also).

Table of Contents

Section I

Introduction

Executive Summary

Over the past ten years, open source software has made strong inroads into markets for proprietary software (Torvalds and Diamond, 2001). Through economic theory and analysis, this paper will seek to uncover what effects this change in the software market is having on innovation. In order to accomplish this goal, this paper will look at some of the economic effects generated by open source software, then attempt to apply them to economic theories of innovation. This paper will then conclude by outlining four key economic effects generated by open source activity, whose aggregate effect on innovation in the software market is ambiguous. However, before we attempt to achieve these goals, we must first understand what exactly open source software is.

What is Open Source Software?

In the simplest sense, open source software (OSS) is software that is distributed at marginal cost and whose source code is freely available to be viewed, modified and redistributed (Schmidt and Schnitzler, 2002).[3] In a more complex sense, the term "open source" describes two separate yet related entities, both of which form the basis of open

[3] A software applications "source code" is the human-readable set of instructions that "tell" the computer how to operate. Most modern software applications, in order to function, must have their readable source code ran through a compiler to convert the human readable instructions into machine readable binary code that is near impossible for humans to understand. Most commercial software is shipped only with this cryptic binary code, thus making it nearly impossible to modify the functionality of commercial software (a great intellectual property protection barrier). Open source software changes this by freely giving away its easily readable and modifiable source code.

source software[4]. The first of these is the open source development model, which is a way of developing software based on decentralization, collaboration and reciprocity. It is a system where individuals from across the globe voluntarily contribute patches, enhancements and features to software projects that interest them (Raymond, 1999). Its roots can be traced back to the scientific research organizations of the 1960's and 1970's such as Bell Labs, Xerox Park and the University of California Berkeley where this type of voluntary "code sharing" was quite prevalent (Raymond, 1999).

Today we see this development model employed by a plethora of what have become known as "open source development communities" (OSDCs). These decentralized organizations are effectively non-profit providers of public goods[5], in that they are producing software that they freely distribute over the Internet for all to use with very few restrictions[6]. Examples of these communities include the Linux development community, the creator of Linux, an operating system in direct competition to Microsoft Windows, and the Apache development community, the creator of the Apache web server, an application that hosts a majority of the web sites currently on the Internet (Lerner and Tirole, 2000).

---

[4] I really must give credit to Stephen Mutkoski of the Intellectual Property & Licensing Group at Microsoft for breaking down the concept of open source software into the clear dichotomy of a development model and a licensing model. While writing this paper, I was lucky enough to sit in on Prof. Dan Laster's Advanced Copyright class at the University of Washington Law School where Stephan gave a highly informational guest lecture on the legal issues surrounding open source.

[5] A good is a public good when it exhibits the two key properties of non-rivalry and non-excludability. Non-rivalry means that the good does not demonstrate scarcity, meaning one person's consumption of the good will not affect anyone else's consumption of that good. Non-excludability means that it is difficult or even impossible to prevent someone from using the good. Open source software in fact characterized by both of these properties. It is non-rivalrous because, as it is software and can be reproduced indefinitely at marginal cost, it can never be scarce. It is non-excludable because it is almost always distributed freely and openly. Other examples of public goods include a national defense system and a system of property rights.

[6] We will learn more about these restrictions in section 5 of this paper.

The other main entity that is often referred to by the term "open source" is the open source style of software licensing. Similar to how commercial software firms like Microsoft license software to the public to control how their products are used, the open source licensing style is the way in which OSS developers license their software to the public to restrict how it is used and distributed. There are in fact many different "open source" licenses all of which adhere to the same loose set of principles. Some of these principles include openly viewable and modifiable source code and free redistribution (Fink, 2003).[7]

Another important licensing term, which is not always stipulated, is that of reciprocity. The reciprocity term states that any derivative work of a piece of code licensed under this term must, when distributed, use the same legal terms as the original piece of code. An example of this would be if an OSS developer wrote a piece of code that would draw circles, licensed it under an open source license that included the reciprocity term, and then put it on the Internet. If then another developer came along, found this code and integrated it into a graphics program she was working on, she would then be obligated by law to license her program under the same terms as the code that drew the circles – thereby self-perpetuating the licensing terms (Gomulkiewicz, 1999).

It is in fact this idea of reciprocity that splits the currently available set of open source licenses into two categories – those that have the reciprocity term and those that do not. The two main licenses from each side of this divide include the GNU Public License (GPL), which forms the model for the reciprocity-based open source licenses and the Berkeley Software Distribution License (BSD), which does the same for the non-

[7] For a more detailed discussion regarding the various key principles of open source licenses, please refer to section 5 of this paper.

reciprocity or “liberal” license type. Both of these license types will be examined in much greater detail in the subsequent sections of this paper (Fink, 2003).

Paper Roadmap

This paper is divided into four main sections. The first of these sections will discuss the history of OSS to gain an understanding of its origins and also set some of the foundations for the rest of the paper. The second section will explore some of the economic effects generated by OSS by looking at such areas as the motivations driving OSS developers, the formation of open source software development communities (OSDCs), and the changes in the market share of commercial software driven by OSS. The third section will take the economic analysis of the previous section one step further by examining the effects that OSS may have on the level of innovation in the software market. The fourth and last section of this paper will conclude by discussing the legal institutions supporting OSS and also by outlining some possible policy suggestions to help encourage the creation of OSS.

Section II

A Brief History of Open Source Software

Introduction

In this section we will trace the origins of the current open source development model to the late 1960's at such scientific research institutions as Bell Labs and MIT. This section will also show how many of the current institutions that now support the creation of open source software (OSS), such as the Berkeley Software Distribution License (BSD) and the GNU Public License (GPL) came into existence and usage.

A Brief History of Open Source Software

The open source software development model, or the activity of openly sharing source code in a collaborative environment, is not new. Its origins can be traced back to the pivotal year of 1969, which marked both the creation of the Unix operating system and the ARPAnet (Advanced Research Projects Agency Network, the predecessor to today's modern Internet). However, it is important to note that prior to Unix there was a preceding operating system named Multics or Multiplexed Information and Computing Service. Multics was a highly ambitious project being jointly sponsored between MIT, Bell Labs (The research and development branch of AT&T) and General Electric (Weber, 2004).

On account of Multics' overreaching goals and a lack of consensus on the project's direction between its sponsors, the project was eventually canceled in 1969 after five years of work. The efforts on Multics, however, were not a complete loss. Two

programmers from Bell Labs, Ken Thompson and Dennis Ritchie, who had worked on Multics, walked away with some important lessons learned; lessons such as “build small neat things instead of grandiose ones”, which would eventually prove useful in the creation of a new, simpler operating system (Weber, 2004).

With a lack of direction at Bell Labs after the collapse of Multics, Ken Thompson took it upon himself, in an astonishing four week period, to write that new, simpler operating system, which he playfully named UNICS or Uniplexed Information and Computing Service. It would eventually be changed to the name that we know it by today, Unix (Weber, 2004).

The usage of Unix grew slowly. By early 1973 there were only about sixteen installations and all within the confines of Bell Labs. However, this changed dramatically after Ken Thompson and Dennis Ritchie jointly presented a paper on Unix at the Association for Computing Machinery Symposium in October of 1973. Shortly afterwards, Bell Labs was inundated with a multitude of requests for copies of Unix (Weber, 2004).

This would have typically been a fantastic business opportunity for AT&T (the parent corporation of Bell Labs). However AT&T was legally bound as a government mandated monopoly to only sell products relating to common carrier activities and Unix, as AT&T’s attorneys reasoned, was not within this product sphere. As a result, AT&T decided to license out Unix at marginal cost and under minimal licensing terms. On account of the resulting low price and strong demand for Unix, it spread quickly across the globe to both universities and research institutions and later to commercial organizations (Weber, 2004).

This situation demonstrated two highly important points. First, Unix was distributed in readable source code form rather than unreadable binary machine code and further, the source code was written in Dennis Ritchie's new, easier to understand language, appropriately named "C", as it was the successor to Bell Labs' "B". This meant that it was possible to understand the inner workings of Unix with relative ease (Raymond, 1999). Second, part of Unix's minimal licensing terms was that it was licensed "as is" and was not accompanied by any type of warranty, service or support.[8] These two factors coupled with the collaborative and open sprit of scientific research that was prevalent at the research oriented organizations which licensed Unix, gave way to a system of collaborative code sharing in which the students, professors and researchers using Unix around the globe began to slowly work together in making (and most importantly, sharing) enhancements, modifications and bug fixes (Weber, 2004). This collaborative movement was then propelled further in 1976 when the UUCP or Unix-to-Unix Copy Program was written. This program enabled Unix users to harness the power of the ARPAnet[9] to share code and communicate across this nascent, global, digital network (Raymond, 1999).

By the late 1970's Unix was becoming very popular and with this growth in popularity, the mindset of AT&T's management was beginning to shift. When AT&T released Unix version 7 in 1979, management realized its true commercial potential and

[8] It is important to point out that both of these licensing terms, distribution at marginal cost and no warranties, are in fact at the core of today's open source software licenses. For a more detailed discussion on this topic, please refer to section 5 of this paper.

[9] The ARPAnet or Advanced Research Projects Agency Network was a project initiated by the United States' Department of Defense in early 1960's to create a global computer network to help link research centers from across the United States at such universities as UCLA and Stanford via a new method of data transfer known as packet switching. What is so important about the ARPAnet is that it set the infrastructural and conceptual framework for what today is the modern Internet.

for the first time decided not to release Unix's source code. The plan to fully commercialize Unix was followed through in 1983. As a result of AT&T's deregulation, AT&T was no longer legally bound to give Unix away at cost and could now sell licenses for a market price (which ran as high as $250,000) under a new division of AT&T named Unix Systems Laboratory. AT&T's shift to a closed, cathedral-like[10] development model effectively halted its role in the collaborative development process that it had helped to establish (Weber, 2004).

At about the same time a programmer named Richard Stallman who worked in the Artificial Intelligence Laboratory at MIT, was starting to become increasingly distraught by AT&T as well as other software firms as they shifted away from releasing their code freely and moved to a more "closed-source" and commercial business model. This movement frustrated Stallman to the point that, in backlash, he founded The Free Software Foundation (FSF) to promote what he coined "free" software or software that has openly viewable and modifiable source code (von Hippel and Krogh, 2003).[11] One of the most important results of the FSF was its creation of the GNU Public License, a software license used by many open source applications today (Fink, 2003).[12]

In spite of AT&T's movement to commercialization, there were still examples of collaborative development efforts to be found. Prior to AT&T's deregulation, the

[10] Eric Raymond, one of the most pivotal thinkers in the open source movement, coined the term "cathedral" style of software development to describe the way in which commercial software firms construct applications. He uses the metaphor of a cathedral to illustrate how commercial software is carefully planned, centrally controlled and constructed by few. He contrasts this idea with what he calls the "bazaar" style of development, where you have less central planning, but more peer-review and a greater amount of contributors to the project.

[11] It is important to point out that by "free", Stallman is referring to freedom, as in the freedom to modify software. Nevertheless, most "free" software is also free in terms of price.

[12] The GNU Public License and open source licenses in general will be discussed further in both Section 4 and Section 5 of this paper.

University of California Berkeley had fully embraced the development and enhancement of the Unix operating system. This put Berkeley in a strong position to continue Unix's development via a collaborative development approach (Weber, 1999).

Berkeley had received a copy of Unix version 6 from Ken Thompson while he was there on a teaching sabbatical. In 1976 two Berkeley graduate students, Chuck Haley and Bill Joy, added some key enhancements to this version of Unix, calling this new version Berkeley Software Distribution or BSD. These enhancements, which became quite popular in the Unix community, included an improved Pascal interpreter (a structured programming language similar to "C") and a useful text editor named "ex" (which we now know this today as "vi") as well as some important performance improvements. Staying in the sprit of the collaborative culture established by Bell Labs, Haley and Joy freely gave their source code to all those who had interest (Weber, 2004).

This was the beginning of a series of events that would help establish Berkeley as a dominant force in the Unix community. By 1983 there was a clear divergence in the development of Unix. AT&T went the commercial, proprietary route while, Berkeley in contrast, stayed on course in the established collaborative development path. This resulted in an interesting situation in which Berkeley's 3BSD (BSD Unix version 3) was chosen by the Department of Defense to be the common operating system to run on all of the ARPAnet's main computers in order to implement a new and enhanced communications standard known as TCP/IP. The Department of Defense chose 3BSD over AT&T's Unix and Digital Equipment's VMS (the other major competing operating system to Unix at the time) on account of the one key feature that made BSD unique, its source code was freely available (Weber, 2004).

With the implementation of the TCP/IP networking standard integrated into the newest version of BSD (4.2BSD), BSD jumped to forefront of the Unix sphere. There was, however, one major caveat to all this, a large majority of the code that made up BSD was created by AT&T and thus subject to its distribution license. With AT&T charging substantial prices for its product, users of BSD were now subject to this new pricing scheme. In order to get around this legal complication, Berkeley undertook the ambitious task, via the collaborative, voluntary development model, of rewriting all of the original AT&T code, thereby creating a completely "clean" version of BSD. By doing this, BSD was no longer infringing AT&T's copyright on Unix and thus not restrained by AT&T's commercial software license (Weber, 2004).

This version of BSD, although never fully completed (it was missing six files), was released under the name Networking Release 2 in 1991 under Berkeley's own, custom software distribution license. The hope was that someone else would at some point complete the missing files (Weber, 2004). The creation of the Berkeley license, later to be known as a BSD-style license, marked an important development in the greater open source software movement, such that this license still is one of the two most common and represents one of two schools of thought on the movement's direction and philosophy (Fink, 2003).[13]

In 1991, a programmer by the name of Bill Jolitz acquired a copy of Berkeley's Networking Release 2 and was able to fulfill Berkeley's hope by completing the missing files while simultaneously porting BSD to the Intel 386 platform. Jolitz then released his

[13] The other school of thought is represented by Richard Stallman's GNU Pubic License (GPL). The GPL contains what is known as a reciprocity clause that effectively self-perpetuates the terms of the software license. In contrast to this, the BSD does not contain this clause and is seen as a more "relaxed" license type. This dichotomy will be discussed in greater detail in both Section 4 and Section 5 of this paper.

finished product, named 386BSD, onto the Internet where it was well received (Raymond, 1999). In fact, the center of the Internet-based, collaborative development movement had shifted yet one more time, moving from Berkeley's BSD to Bill Jolitz's 386BSD. An Internet-based community quickly formed where a high degree of bug fixes and enhancements were being exchanged voluntarily (Weber, 2004).

This ideal open source situation did not last long however, eventually 386BSD forked into a variety of projects, and its influence diminished (Raymond, 1999). However, at the same time 386BSD was being developed, a young Finnish university student named Linus Torvalds began writing his own Unix variant based upon a Dutch university's Unix teaching tool called Minux.[14] Torvalds' project, that later became known as Linux, very quickly gained popularity and developed around it a large, decentralized, voluntary development community that would later form the template of future open source software projects. By 1994 when Linux version 1.0 was released, the Linux development community had firmly established itself as the new center in the modern open source development movement (Torvalds and Diamond, 2001).

Conclusion

In summary, the development style that open source software employs originated in the scientific and academic research organizations of the 1960's and 1970's. The first instance of this development system was seen in the construction and evolution of the Unix operating system. Over the years and through the usage of this decentralized, voluntary development model, Unix transformed and split into many variations. Along

[14] It is interesting to point out that both Jolitz and Torvalds were unknowingly working in parallel on the same goal of writing a variation of Unix to run on the Intel 386 platform. In Torvalds autobiography, he admitted that had Internet technology been better than it was in the early 1990's, he may have learned of the 386BSD project and not embarked upon the creation of Linux. In the end however, Linux prevailed as the 386-based Unix variant of choice and helped drive the diminishing influence of Jolitz's 386BSD.

the way, we saw the creation of various software licenses that would later underpin the open source movement. One of these subsequent Unix variants would eventually inspire a young Finnish graduate student to create his own Unix variation that would ultimately prove to be the quintessential open source application and spark the open source revolution.

Section III

Important Economic Issues of Open Source Software

Introduction

In the previous section we explored the origins of the modern institutions that underpin the current open source movement. The goal of this section is to look at these institutions in their modern form and to try to identify and understand some of the important economic issues and effects that are generated by these institutions. Further, in this modern context, the challenge open source software poses to proprietary software firms has become a strong point of contention between many, including such individuals as information technology managers, computer programming enthusiasts (hackers)[15] and recently, even politicians (Torvalds and Diamond, 2001).

These issues and effects will be explored in three steps. First, I will explain the motivations and incentives driving hackers, who constitute the open source development communities. I will then propose some reasons for the formation of these groups, using such examples as Linux and Apache as guides. And lastly, I will explore some of the economic relationships between proprietary software firms and open source development communities by looking at relevant market research and data.

[15] The term "hacker" in modern parlance has unfortunately attained a rather negative connotation. Today this word is typically associated with anti-social individuals who spend their free time writing destructive code for "fun". Technically the correct word for these individuals is "cracker". So for the sake of clarification, for the rest of this paper when I use the term "hacker", I am simply referring to individuals who greatly enjoy writing code – not angry programmers with a destructive disposition.

Hacker Motivations

The common, oft-asked and enigmatic question posed in such earlier works as Josh Lerner's and Jean Tirole's paper entitled *The Simple Economics of Open Source*[16], of why do thousands of highly skilled computer programmers engage in the seemingly altruistic practice of writing valuable source code only to give it up freely, is a highly complex one. This question is one that embodies a broad mix of philosophical, sociological and economic components. Nevertheless, through my research, three motivational drives seem apparent. These include artistry, belief and most importantly, business need.

For many, writing code is far more than a means to an end or a tool to achieve a task; rather, it is artistry. As Steven Weber puts it in his book *The Success of Open Source*, and as I can attest from my experience having operated in this realm, programmers never refer to code in terms of efficiency or other such sterile descriptors, rather, adjectives such as "ugly", "clean" or "beautiful" are the common parlance. Programming is a form of expression, a form of self-identity and along with that comes great pride in one's craft. Linus Torvalds further expresses this thought in his autobiography *Just for Fun*:

> It's Art with a capital A. It's the Mona Lisa, but it's also the end result of a long night of programming, and it's an end result that you as a programmer are damned *proud* of. It's something so precious that selling it isn't even possible: It's indelibly a part of who you are. (Torvalds and Diamond, 2001)

In this same line of thought, programmers also derive joy from programming; it is an artistic task that is also intellectually stimulating and fun. This was confirmed by the

[16] In this paper, among other things, Learner and Tirole apply theories from the field of labor economics to unravel the motivations of open source software developers.

Boston Consulting Group's 2001 survey of open source developer motivations, where when volunteers had a choice between eleven main motivational factors, "Intellectual Stimulation" was one of the two most popular at 43.2%[17]. What this survey shows is that there is a high degree of satisfaction to be gained simply from the creation of code, and that satisfaction alone acts as a powerful motivator in the engagement of open source development communities.

Another key motivational factor, which dates back to the origins of the collaborative development movement during the 1960's at such institutions as MIT and Berkeley, is one of personal ideology. This ideology is the belief that all code should be freely shared. Torvalds again augments this point in his autobiography:

> What started out in my messy Helsinki bedroom has grown to become the largest collaborative development project in the history of the world. It began as an ideology shared by software developers who believed that computer code should be shared freely (Torvalds and Diamond, 2001)

The Boston Consulting Groups Survey also confirms this point, in which out of the same eleven options of motivational factors, the response "Code should be open" was ranked third at 34.2%.

In Eric Raymond's essay *The Magic Cauldron*, he attempts to use "a hard-nosed economic explanation" of what drives hackers to freely write software enhancements and then contribute them back to the open source community. In short, he effectively argues that many hackers do this on the clear basis of business need. In the case of open source software, the classic economic model of producers and consumers is becoming quite blurred in that now single individuals embody both of these roles[18]. The users of open

[17] It turns out that the choice of "Intellectual Stimulation" was in fact tied exactly with "Improves Skill" at 43.2% as the number one and number two most popular choices in the Boston Consulting Group's survey.
[18] To further augment this point – perhaps what we are seeing here within the realm of open source software development is in fact a breakdown of the Fisher Separation Theorem. The Fisher Separation

source software are also the ones producing it. On account of this, we see a rather interesting dynamic of reciprocity taking place.

Raymond argues that in terms of what drives hackers to write enhancements and simply not employ a free-rider approach of just waiting for wanted code to magically spring forth from the community is that hackers, as users of the software for often mission critical tasks at their places of employment, do not just need solutions but they need them in a short order (Raymond, 1999). As Raymond explains in his essay:

> It's seldom possible to predict when someone else will finish a given piece of needed work. If the payoff from fixing a bug or adding a feature is sufficient to any potential contributor, that person will dive in and do it (at which point the fact that everyone else is a free rider becomes irrelevant) (Raymond, 1999).

The fact that it is risky to wait for a needed feature of bug fix, explains why hackers make such modifications. But the question of why don't hackers keep their enhancement to themselves is still outstanding. To answer this, Raymond employs a game-theory analysis which suggests that the payoff is higher to submit the patch back to the community, because if the individual were to decide not to, they would be forced to "incur a future cost – the effort involved in re-merging the patch into the source base in each new release" (Raymond, 1999). Thus, by submitting the patch to the community pool, costs are reduced to the individual and interestingly, to the community as well.

Some have argued that the open source software is an ideological movement to "beat proprietary software". However, according to the Boston Consulting Group's Survey, this is not a significant factor in the usage of and contribution to open source software. Of the eleven options offered to the survey participants, this choice was ranked last at 11.3%. This helps to dispel the myth that the open source software movement is

---

Theorem states that specialization in markets will lead to the separate and distinct roles of both producers and consumers.

nothing but an ideological backlash against the “tyranny” of commercial, closed source software.

In the end, what is important to understand from the three identified motivational drives of artistry, belief and business need is that the common link between them is self-interest – not altruism. If either to maximize utility through the creation of beautiful code, the exercising of beliefs, or simply taking part in this movement because it is a tool of business cost minimization, these are all activities that are in the best self-interest of the individual. Perhaps Adam Smith would agree that among the butcher, the baker and brewer in his most quoted piece of economic prose, we could also add the hacker (Schmidt and Schnitzler, 2002).[19]

Open Source Community Formation

What drives these open source communities to form in the first place is yet another complex question, where each instance of community formation seems to embody its own unique set of characteristics. However, in looking at the origins of the two strongest open source communities, Linux and Apache, a theme does seem to emerge, one that encapsulates all of the aforementioned motivational drives.

Linus Torvalds, the originator of Linux, was a computer science student at the University of Helsinki and simply wanted to be able to work from home on his school work and avoid the long lines for computer access at his university campus (Weber, 2000). This, however, was quite problematic in that his school used Unix based machines and at the time there were very few Unix operating system variants offered for

[19] In reference to the following quote from Adam Smith’s *The Wealth of Nations*: "It is not from the benevolence of the butcher, the brewer, or the baker that we expect our dinner, but from their regard to their own interest. We address ourselves, not to their humanity but to their self-love, and never talk to them of our own necessities but of their advantages."

personal computer hardware. The ones that were available were all quite lacking in features. On account of this, Torvalds took it upon himself, in an act of inspired industriousness, to write his own Unix based operating system for personal computer hardware. After getting a good start on the project and posting his code on the Internet, a flood of interest poured in. This led the formation of what is now the largest open source development community in existence (Torvalds and Diamond, 2001).

The development of Apache had a similar genesis; Brian Behlendorf was a programmer working on *Wired* magazine's parent company's web site known as HotWired. He, like many other web programmers at the time used the standard web server application known as NCSA HTTPd which was developed at the National Center for Supercomputer Applications at the University of Illinois. However, over time, the NCSA reduced support for their product, making it difficult for Behlendorf, as well as others using the product to have their evolving needs met. So, in yet another act of inspired industriousness, Behlendorf took it upon himself, along with a few cohorts using the web server, to simply build their own server application, which then they would be able to freely to enhance as needed. This then led to the creation of the Apache development community (Lerner and Tirole, 2000).

The important lesson from these two distinct situations is that both Torvalds and Behlendorf, as consumers of computer software, needed a product that did not exist on the market. In creating it themselves, other consumers with similar demand characteristics used their products and joined their respective development communities, demonstrating that there was indeed a market for these goods. One possible economic explanation for this situation of independent development is that the cost posed to profit-

maximizing firms of meeting these unique demand characteristics with an enhanced or new product was greater than the collectable value of that product in the market. As a consequence, profit-maximizing firms did not go forth with development in spite of a market demand. This then, it seems, led to the formation of the aforementioned open source development communities. Similar situations and incentives may in fact explain the emergence of many other such open source development communities.

Economic Effects

In Martin Fink's book *The Business and Economics of Linux and Open Source*, he makes a key point that given time, the value of proprietary software eventually decreases through a process of "devaluation [which] comes from competitive forces and a somewhat natural commodity effect". In the technologically dynamic market of software we see a natural process of Joseph Schumpeter's "Creative Destruction", where through innovation, one product eventually gives way to another (Fink, 2003; Schumpeter, 1942). This process however, given such barriers to entry as substantial upfront sunk costs, high switching costs and network externalities, is at times a rather slow and unpredictable process.[20] However, as Fink again points out, the introduction of open source development communities may in fact speed up this devaluation process due to the

[20] In looking at each of these barriers, the cost of hiring the highly skilled labor to produce software coupled with the substantial time it takes to actually complete and properly test a software project before it can be released onto the market leads to its high upfront costs. These costs are sunk because it is often quite difficult to repurpose code in other projects once it has been written, this is in spite of attempts in modern programming design to minimize this through the employment of such techniques as modularization and object oriented programming. We see high switching costs both from a user perspective in that it takes time and effort to learn how to use a new "system" and from an infrastructural standpoint in that once an interdependent system of software has been installed, it is often costly and difficult to switch out one of its components. Lastly we see network externalities, most notably in the desktop market, on account of the fact that as more users choose one platform, that platform becomes more valuable to all users, thus sometimes pushing out entry from competition.

inherit economic and organizational advantages to proprietary software firms. This concept is illustrated below in figure 3-1.

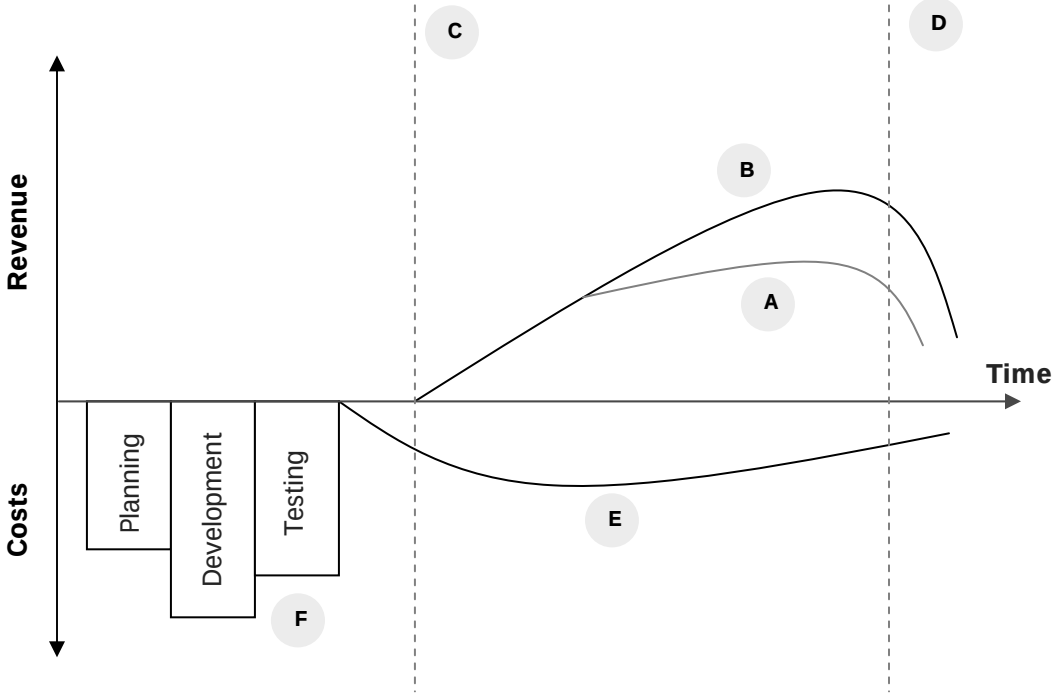


Figure 3-1: An illustration of the accelerated devaluation process generated by the introduction of an open source alternative to a piece of proprietary software

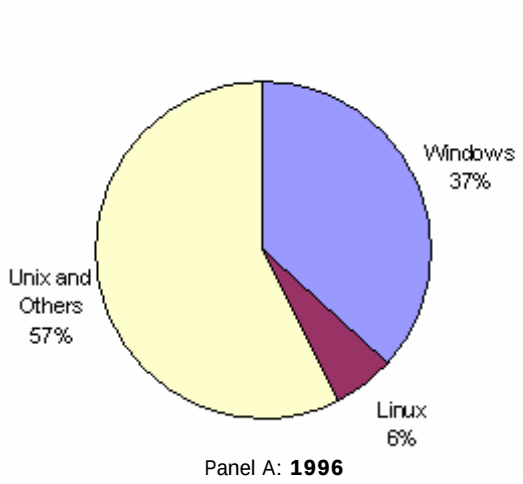
This graph demonstrates the impact that the introduction of an open source alternative would have on the future revenue stream of the competing proprietary software firm. Point **(A)** on the graph is the new revenue curve of the proprietary firm after the introduction of the open source alternative. Point **(B)** is the original revenue curve. Point **(C)** is the point in time in which the proprietary product becomes available on the market. Point **(D)** is the point in time in which the product experiences obsolescence, and thus, revenue decreases. Point **(E)** is the marginal cost of producing each unit (includes production, marketing, support, et cetera). Point **(F)** is the upfront fixed sunk cost.

Source: The Business and Economics of Linux and Open Source by Martin Fink

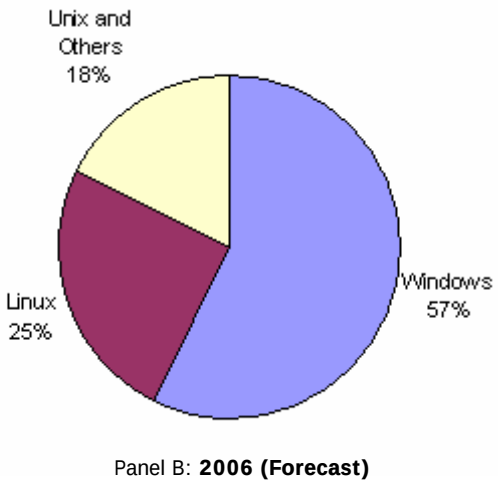
The first of these advantages is that fact that open source development communities do not face the same upfront sunk fixed costs as proprietary firms. This is possible because these developers, as we learned earlier, have non-monetary motivations. For this same reason, their second advantage is that they distribute their product at

marginal cost, which in the case of software, is zero.[21] Lastly, through the large numbers of developers working collaboratively across open source communities, the time it takes to actually write the software is reduced. Effectively, through this accelerated devaluation process, the market share of the proprietary software is being challenged as well as its revenue stream on account of the open source contender, and in fact, there is evidence which demonstrates this dynamic.

Regarding the shifts in market share, in 1996 International Data Corporation (IDC) estimated the market share of Linux in the server operating system market at 6%. In 2003 IDC forecasted that by 2006 Linux's market share would jump to 25%. This is demonstrated by the two graphs in figure 3-2 below:



Panel A: **1996**



Panel B: **2006 (Forecast)**

Figure 3-2: Transformation of market structure in the server operating system market

In this graph we see how between 1996 and 2006 there was a rather large restructuring of market share in the server operating system market. The most notable change is the growth of Linux's influence, growing from a mere 6% in 1996 to a forecasted 25% in 2006.

Source: International Data Corporation

[21] It is of course completely feasible for profit-maximizing software firms to distribute their software at marginal cost, however, if they were to do this, they would never be able to recoup the sunk fixed cost of developing the software thus halting development of subsequent applications.

Another notable issue shown in the figure above is the concurrent growth of Windows' influence along with Linux's. In spite of Linux eroding the market share of proprietary fringe server operating systems, Microsoft's Windows has continued to grow. However, this growth is tempered by two pieces of data. The first is the fact that in 2005 IDC estimated Window's market share at 58%, therefore, IDC's 2006 forecast reflects their estimate that Windows will lose 1% of market share over this time period. Second, a study by the Gartner Group in 2003 predicted that between 2003 and 2008 the growth rate of Linux would be 20.7% – four times that of Windows. From these two pieces of data, it becomes clear that Linux is at least posing a strong challenge to Windows' position.

Turning our attention to evidence of revenue reduction, in November of 1998, a series of internal Microsoft memos regarding their view on the growth of Linux were leaked by a former employee. These memos, later to be known as the Halloween memos, illustrated that Linux, in the eyes of Microsoft, in fact posed a rather large threat to the dominance of Windows, especially in the server market. One of the most interesting quotes taken from these memos was one from an engineer named Vinod Valloppillil who stated that Linux poses a "significant near-term revenue threat to Windows NT Server". In their official response, Microsoft of course denied this; however, it is nevertheless a rather compelling statement (Shankland, 1998; Weber, 2004).

Another piece of evidence indicating the possibility of a future revenue reduction comes from a 2003 report by Forrester Research, in which 81% of the IT managers and executives surveyed believed that open source software will to some degree affect

Microsoft, and of those 81%, the most popular answer of four options on how this would take place was that Microsoft would in fact result to reducing prices.

Conclusion

In summary, from this analysis, a few of key points have emerged. First, contributors to open source software are typically motivated not by destructive forces or monetary gain, but rather by a combination of utility maximization and cost minimization, both driven by self-interest. Second, there are a number of highly sophisticated users, whose demand characteristics are too costly for profit-maximizing firms to meet. This leads these developers to collaborate together and collectively create the products they need. Third, on account of open source software communities' inherit economic advantages, they are able to enter the software market with greater ease and thereby accelerate the devaluation process of proprietary software. This devaluation process should then result in an erosion of both market share and future expected revenue of proprietary software firms.

Section IV

Open Source Software & Innovation

Introduction

In the pervious section we saw the likely impact that open source development communities (OSDCs) have on both market structure and future expected revenue of commercial software firms. In this section, we will take this analysis one step further by exploring how the exogenous changes generated by open source affect the software market as well as other unique open source issues affect innovation and incentives to innovate.

Open Source & Innovation: Market Structure

As we saw in the previous section, open source development communities are restructuring the software market in such a way that market shares are becoming more equalized. In the case of Linux, commercial software firms, once dominant in the server market, are now increasingly becoming on equal footing with Linux as Linux's market share continues to gain ground against them. Further, due to Linux's growth, it is acting as a bolster against the growing influence of Microsoft. In the case of Apache, since its introduction in 1994, it has held a roughly dominant position but has still shared the market with other proprietary web servers, most notably, Microsoft's IIS. Thus in essence, we have one OSDC that is eroding market share from commercial firms and another that is maintaining its position in market share against fringe firms. The next step is to ask how these changes in market structure are affecting innovation.

Over the years, there has been a good deal of literature written linking market structure and innovation. One of the earliest and most important of these is Joseph Schumpeter's *Capitalism, Socialism and Democracy*. In this book, Schumpeter asserts that in a market with a large degree of technological change, highly concentrated markets are best suited to innovate. He contends that large firms who employ monopoly power and charge a price high above marginal cost are able then to allocate these rents to research and development purposes. This inflow to R&D should then lead to market innovations not seen in highly competitive industries (Schumpeter, 1942).

In the case of software, this analysis is highly relevant. On account of the dual factors that software requires significant upfront, sunk R&D costs and that software's marginal cost is zero, it would be unlikely to see high levels of innovation arising from this industry unless software firms have the ability to charge a price above marginal cost. This is because without the ability to collect post-development quasi-rents, there would be no way for software firms to recoup their sunk fixed costs, thus stunting innovation. This is on account of the fact that, as the literature on the economics of innovation states, firms will not engage in an innovation unless the expected future collectable value is at least equal to its costs (Scotchmer, 2004).

From this we can infer that a market with a structure of perfect competition should not produce nearly as many innovations as a market with a degree of concentration (Scherer, 1980). The question is simply how much concentration is ideal. If there were such a correlation that the higher the level of concentration, the greater the innovation level, then OSDCs would actually pose a threat to market innovation on account of their market concentration deluding behavior. However if instead, a balance

of market share between fringe and dominate firms led to optimal innovation, then OSDCs would indeed be making a positive contribution to innovation. And it is just this type of market structure that F. M. Scherer in his book *Industrial Market Structure and Economic Performance* argues is indeed optimal to produce market innovations. As Scherer puts it in his conclusion to chapter 15:

> What is needed for rapid technical progress is a subtle blend of competition and monopoly, with more emphasis in general on the former than the latter, and with the role of monopolistic elements diminishing when rich technological opportunities exist (Scherer, 1980).

This relationship is further demonstrated below in figure 4-1:

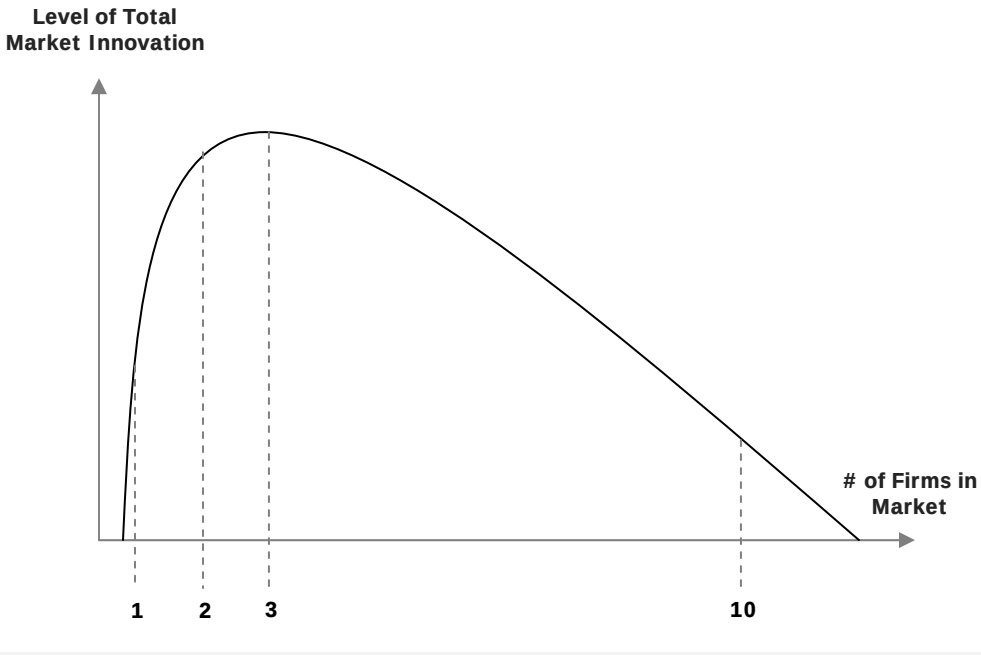


Figure 4-1: Correlation between market structure (concentration of the market) and total level of market innovation

In F. M. Scherer's book, *Industrial Market Structure and Economic Performance*, a relationship between innovation and market concentration is described such that innovation is highest when a market contains monopolistic elements, but is not completely concentrated. Further, it is discussed that the level of market innovation deteriorates as the concentration of the market deludes, ceteris paribus. This relationship is illustrated in the figure above.

To turn to an empirical analysis on this matter, in a paper by Zoltan Acs and David Audretsch entitled *Innovation, Market Structure, and Firm Size,* Acs and

Audretsch conducted an empirical study trying to find the relationship between firm size and innovation level. Through their data regressions and analysis, Acs and Audretsch reached two main conclusions in their paper. The first was that the Schumpeterian hypothesis that large firms are best suited for innovating needs to be augmented to include the type of industry being studied, for depending on different industry characteristics, this hypothesis does not always hold. In taking this idea to the next level, Acs and Audretsch's second conclusion was that large firms have the "innovative advantage" in industries that are highly capital intensive and unionized while small firms are best at innovation in industries where skilled labor is highly employed and large firms dominate the market share landscape. It is important to point out that in fact the software industry matches that latter of these two industry types rather well. In closing, Acs and Audretsch make the very interesting observation that:

> At least for these industries [referring to the latter of the two described above], the conclusion of Scherer (1980) that markets composed of a diversity of firm sizes are perhaps the most conducive to innovation activity is reinforced (Acs and Audretsch, 1987).

Therefore, from both a theoretical and empirical standpoint, it appears that the market structure transformation created by the entry of OSDCs appears to be positive in terms of creating an environment conducive to market innovation (Schmidt and Schnitzler, 2002).

Open Source & Innovation: Open Source as an Open Standard

One of the most interesting and more recent developments in the open source movement has been the surge of commercial firms embracing open source and using it as a part of their respective business models. Examples of this include IBM, who now uses Linux and Apache at the core of its enterprise business solutions, Covalent, who builds

proprietary enhancements to Apache for enterprise business needs, and Lineo, who develops embedded Linux devices (Fink, 2003).[22]

What is important about business models such as these is that they are using open source applications as development platforms to create more advanced applications and application enhancements.[23] It is of course technologically and legally feasible to use proprietary software as a platform for development. However, there are two important constraints. First, using proprietary software as a development platform means locking yourself into a proprietary standard, becoming completely beholden to the owner of the technology. Second, proprietary technology comes with licensing fees, controlled by the owner, for the usage of the owner's intellectual property. For these two reasons, open source becomes a rather enticing alternative as an open development platform.

Interestingly, it appears that the literature on the economics of innovation concurs with this analysis. In Suzanne Scotchmer's book *Innovation and Incentives*, she describes three types of cumulative innovation. The first is a "basic" innovation that has the possibility of leading to multiple subsequent innovations. The second is a "multiple input" innovation that requires more than one prior innovation to come into existence. The last is one that sits on what Scotchmer describes as a "quality ladder" and competes with successive innovations. It is the first of these that maps to our discussion on open source software, in that applications such as Linux and Apache set the foundation for

[22] Embedded software devices are physical "machines" such as a microwaves or even automobiles that contain software applications permanently installed within its hardware. This type of software is often called "firmware".

[23] The term "platform" often refers to a type of operating system software combined with a supporting set of hardware, however, in this context I am using this term in a more metaphorical sense such that it is synonymous with simply a "springboard" for subsequent development.

multiple subsequent innovations such as those created by firms like Covalent and Lineo (Scotchmer, 2004).

Scotchmer goes on to explain that the key factor regarding incentives to innovate in subsequent innovations is that firms "will not invest unless the difference between the value of the innovation and the cost [of the license of the prior innovation] are relatively high". In the case of open source, this difference should often be high (so long as the collectable value of the innovation is high) due to the fact that the cost of the ex ante license should always be zero. Thus, open source as an open development platform standard may lead to a high level of cumulative innovations (Scotchmer, 2004).

Open Source & Innovation: GNU Public License

So far we have looked at ways in which the emergence of OSDCs are creating an environment conducive to market innovations, however, we will now switch gears and look at some ways in which the growth of open source development may detract from creating market conditions compatible to innovation incentives.

There are many types of open source distribution licenses, however most fall into either one of two groups. The first is known as the Berkeley Software Distribution License (BSD) or "liberal" license type. This license type is for the most part quite innocuous in that any derivative work based on a piece of BSD code has no further legal obligations. In older versions of the BSD, derivative works had to place a note giving credit to UC Berkeley in the subsequent licenses, however, now this stipulation has been removed. The key unique feature of this license type is that any derivative work based on BSD code has the freedom to show or hide its source when distributed, thus giving the option for the secondary work to be used commercially (Weber, 2004).

The second is known as the GNU Public License (GPL) or "viral" license type. It is this license that is the most important to discuss. This license was created in 1983 by the MIT programmer Richard Stallman out of the belief that source code should always be freely viewable and modifiable in order to help foster the creation of collaborative software communities. In his own words:

> The idea that the proprietary software social system – the system that says you are not allowed to share or change software – is unsocial, that it is unethical, that it is simply wrong may come as a surprise to some people. But what else can we say about a system based on dividing the public and keeping users helpless (Stallman, 1999)?

The main feature of the GPL is that it stipulates that all subsequent developments on a piece of code distributed under the GPL must make viewable its source code in all of its distributions and license them under the same terms – hence the term "viral". Because the bulk of the value of any software distribution is in the source code, and because it is stipulated that developers of the derivative work must release their source code freely, they are effectively being forced to forego a majority of their rents. They then are only able to collect value on complementary goods such as media or support material (Fink, 2003).

However, it is important to point out that by "derivative work" I am referring only to a subsequent program whose code intermingles with the GPL'ed code. Thus it is quite possible to write software that interacts with GPL'ed code but lives side-by-side with the licensed code. However, in the case that a programmer would want to enhance the functionality of a piece of GPL'ed code, a modification of the licensed code may be necessary, in which case the situation would become rather precarious. Nevertheless, there are ways to work around this. As Martin Fink explains in his book *The Business and Economics of Linux and Open Source*, it is quite possible to separate out the portion

of the derivative code that intermingles with the GPL'ed code from the rest of the new program. The programmer can then license each portion separately such that the intermingled portion would need to be licensed under the GPL while other section would have the freedom to be distributed under a commercial license (Fink, 2003).

Going back to the question of innovation, from my research there appears to be one major issue in which the GPL may indeed pose a threat to innovation in the software market – this is the issue of legal risk. As we learned, it is indeed possible to build proprietary enhancements to GPL'ed code – it just requires a bit of finagling. The main questions then become: are we dealing with a situation of asymmetric information and are these legal restrictions of the GPL a disincentive to cumulative innovation? This in fact may be so, as Fink makes the related point in his book that "[t]here are many individuals who believe that you cannot sell or run commercial software on [GPL'ed software]". If there are many individuals who believe this, then it is easy to imagine how many do not understand that it is possible to augment GPL'ed software via the separation of the enhancement into two licensing schemes. Thus, the cost of learning this information either through time or the advice of legal counsel coupled with the legal risk of possibly not separating the code properly may in fact act as a powerful disincentive to cumulative innovation by raising its cost. This then counters the earlier point that open source as an open standard may lead to further cumulative innovation on account of the lack of interest in embracing that standard due to its increased cost.

It is important to point out that the risk of infringing the GPL indeed has real consequences. In April of 2004, a German court ruled in favor of the GPL, giving it its first ever legal precedent. The plaintiff involved in the case was a German programmer

named Harald Welte who was a main author of a piece of GPL'ed code called "netfilter". Welte sued the Dutch company Sitecom, alleging they were using his code in one of their applications while not adhering to the restrictions of the GPL by not releasing their source code freely. In its ruling, the court granted Welte's request for a preliminary injunction stopping Sitecom from distributing its product without its source code. Sitecom now complies with the GPL and distributes its source code freely.[24] Although this was in Germany and not the United States, and hence not the death knell to American GPL infringers, the case does demonstrates the risks of breaking the GPL and that such risks need to be considered before engaging in augmenting a piece of GPL'ed code (Shankland, 2004).

Conclusion

In answering the question how innovation and incentives to innovate are affected by open source, there are no perfectly clear conclusions. We have learned that the affect OSDCs have on market structure creates an environment conducive to innovation, and further that open source applications such as Apache can be viewed as an open standard compatible to the development of cumulative innovations. However, we have also seen how open source licenses such as the GPL pose risks to innovators that may increase their development costs compared to more liberal license styles, and thus stunt innovation.

[24] Although, Welte contends that Sitecom is not freely releasing their complete source code, only selective sections (Shankland, 2004).

Section V

Open Source Software Legal Institutions & Policy

Introduction

In the pervious section we learned how open source software (OSS) development and open source software development communities (OSDCs) may help to create an environment compatible to innovation in the software market.[25] In this section we will explore what legal institutions sustain and propel open source development so that we will have the foundation to explore what possible policies may help allow for the encouragement of open source development.

Open Source Legal Institutions

The key legal institutions that allow for the sustainability of open source software development are copyright and contract law (Gomulkiewicz, 1999). Together, these institutions allow for the main vehicle of OSS development, mass-market software licenses (Gomulkiewicz, 1999). In the previous section we introduced the GPL and BSD, which are both examples of mass-market software licenses and are both central to OSS development.

Similar to commercial software firms, OSS developers choose to use licensing schemes rather than placing their code into the public domain because it allows the developer to state the terms in which the recipient of the open source code can use that code. For example, because many OSS developers would not like to see their code end

[25] As discussed in the previous section, this is assuming that the idiosyncrasies of employing the GPL become widely understood.

up in a proprietary product, they can use a open source mass-market license design that states that any future derivative work of the code covered by the license must expose its source in a readable format, therefore greatly discouraging a proprietary software firm from using that code (Fink, 2003). And as we learned earlier, one of the most popular mass-market licenses, the GPL, accomplishes just that.

Open source mass-market licenses derive their power from the terms stated within them (Gomulkiewicz, 1999). These terms dictate precisely what a recipient of the licensed code can and cannot do (Gomulkiewicz, 1999). In the case of proprietary software licenses, some of the typical terms include that a recipient cannot modify the program or redistribute it (Fink, 2003). On the other hand, open source mass-market software licenses state very different terms and in fact, an organization known as the Open Source Initiative (OSI) has created a framework known as the Open Source Definition (OSD) of about ten key licensing terms that if employed in a license, will designate the licensed code as "open source" (Fink, 2003). Four key principles embodied in these terms include:

- **Free Redistribution**: The recipient of the program can give it away to anybody (Fink, 2003).
- **Open Source Code**: The program must expose its source code in a readable and modifiable format (Fink, 2003).
- **Allowed Derived Works**: The recipient must have the ability to make and distribute a derivate program from the licensed code and be able to use the same licensing terms (Fink, 2003).
- **No Liability**: The licensor cannot be liable for any damages the licensed code may cause a recipient and thus, no warranties accompany the code. This is an important point that we will come back to (Gomulkiewicz, 1999).

It is important to note that the self-perpetuating clause, which gives the GPL its strength, is not among these terms. It turns out there is yet another open source licensing standards group, the Free Software Foundation (FSF), not surprisingly headed by the GPL's creator, Richard Stallman. The FSF's competing framework to the OSI's is in fact quite similar, except for the key difference that it requires the self-perpetuating clause to be present in the license (Fink, 2003). Once this happens, the license can be called "copylefted" (Fink, 2003). The impetus of this extra requirement goes back to the philosophical beliefs of Stallman discussed in the previous section (Weber, 2004).

Policy Suggestions

Now that we have seen the legal institutions underpinning open source development, we are prepared to shift our focus to what policy levers these institutions expose and how they could be manipulated to encourage open source development.[26]

In Robert Gomulkiewicz's paper *How Copyleft Uses License Rights to Succeed in the Open Source Revolution*, he makes the key point that the open source licensing principle of "No Liability" is paramount to the success of OSS development. The fact that a developer of open source code has the ability to distribute her work accompanied by no warranties effectively shifts the risk from the licensor of the code to the recipient of the code (Gomulkiewicz, 1999). This is a highly important point as Gomulkiewicz explains in his paper:

> Valid reasons underlie this risk-shifting strategy…individual hackers are unwilling to assume the risk of a multi-million dollar class action law suit as the consequences of pursuing their passion for hacking code. "Low Risk" also means low barriers to entry; anyone can contribute code to the process, not just those that can afford insurance or lawyers… (Gomulkiewicz, 1999)

[26] It is important to point out that in spite of outlining these policy suggestions to encourage open source development, I am not advocating that these policies should be implemented. This section is solely to provide an illustration on the inner workings of open source legal institutions.

Thus we can infer that if OSS developers were not able to disclaim liability[27] on their code, it would substantially increase open source code's perceived development costs on account of such high legal risks – greatly discouraging OSS development. This is exactly why it is imperative from a policy stand point to not allow any piece of legislation that would make it illegal to disclaim liability in a mass-market software license. The ability to disclaim liability is a key driver to OSS development and therefore it must be protected (Gomulkiewicz, 1999).

In another paper by Robert Gomulkiewicz titled *De-Bugging Open Source Software Licensing*, Gomulkiewicz makes the observation that many of the open source licenses currently in usage are in fact quite flawed and outdated. An example of this includes the GPL which does not state explicitly that the recipient of GPL'ed code can in fact run that code – it does however unambiguously state that she may modify and redistribute it (Gomulkiewicz, 2002). Furthermore, there are in fact over fifty different open source licenses, many of which overlap on the periphery – and no clear guide as which to choose for what purpose (Gomulkiewicz, 2002; Fink, 2003). Thus, with the current confusing offering of flawed licensing schemes and no organization updating or maintaining them, open source developers are being held at a disadvantage.

Gomulkiewicz goes on to propose a solution to this dilemma by the creation of a central open source licensing organization. This organization would be a forum for open source developers and lawyers to work together in updating and maintaining open source licenses to best serve the OSS development community (Gomulkiewicz, 2002). If this

[27] An example of the liability being disclaimed is the liability of having a downstream user of the OSS code being able to put fault on the originator of the OSS code or an upstream developer for any potential problem that may result on account of the code and give cause for legal recourse.

organization were to be created, the costs of open source development facing the individual developer would be decreased by the removal of this logistical barrier. On account of this, the formation of this organization would be an excellent policy to encourage the creation of open source software.

A third and final policy that needs to be considered is one regarding software patents. In 1995 the United States Patent and Trademark Office (PTO) developed for the first time software patent guidelines to reflect all of the previous cases regarding patents on software, thereby cementing that indeed software applications in the United States meet the statutory requirement to become patentable (Tysver, 2000). Furthermore, currently in the European Union, a similar development is taking place. The EU council is deciding whether or not to allow patents on such ethereal items as software and business processes (Perens, 2005).

The main issue surrounding software patents is that they are costly to file and even more expensive to defend in court (Perens, 2005). This high cost puts individual open source developers at a disadvantage to large proprietary software firms because profitable commercial software firms should theoretically be able to accumulate proportionally more software patents than individual open source developers. On account of this, strong enforcement of software patents may in fact stunt OSS development. Despite that fact that this is a highly complex issue and a complete discussion on this topic is out of the scope of this paper, we might infer from this high-level analysis that software patents are in fact not compatible to OSS development, and thus from a policy position, should be enforced more selectively or even not at all.

Conclusion

In summary, the key legal institution supporting open source software development is the mass-market software license, which is based upon both copyright and contract law. More specifically, it is the exact terms stated within these licenses, such as free redistribution and openly viewable source code that propel the open source movement.

Furthermore, there are three possible policies that can be employed to help encourage open source software development. The first is the protection of the ability for an open source developer to disclaim liability from possible damages of their code. The second is the creation of a central open source licensing organization to help manage, maintain and update the currently flawed and enigmatic plethora of open source licenses. The third and last is limiting of the enforcement of software patents, whose presence and usage may discourage open source software development.

Section VI

Conclusion

Final Conclusions

In this paper, we have covered a wide breadth of material – everything from the history of open source software to intellectual property policy suggestions on how to encourage and sustain its development. Despite open source software's sometimes seemingly ethereal qualities, we have attempted to use some economic theories to better understand what open source software is and to answer the pivotal question of whether open source positively contributes to innovation in the software market.

In attempting to answer this question, it appears there are no perfectly clear conclusions to be drawn. Rather, this paper has identified four economic effects generated by open source development activity whose aggregate effect remains unclear. Two of these effects appear to contribute positively to the level of innovation while the other two appear to do the opposite. These effects include:

Positive Effects

- **Market Structure Transformation**: As we saw in sections 3 and 4, the market share data reflected a fundamental transformation between the years of 1996 and 2006 in which the latter market structure, according to the economic literature, should create an environment more conducive to innovation.

- **Open Source as Development Platform**: On account of open source software's unique licensing terms of free redistribution and openly available source code, it seems that open source software lowers the cost of ex ante licenses for cumulative software innovations thus adding positively to the level of innovation in the software market.

Negative Effects

- **Accelerated Software Devaluation Process**: In section 2 we saw how on account of open source development communities unique characteristics, they are able to enter the software market with greater ease, thus helping to accelerate the devaluation process of proprietary software. The threat of this devaluation process may then act as a disincentive to the engagement of new, proprietary innovative projects, which may have been engaged upon otherwise.

- **GPL Risks & Limitations**: As discussed in section 4, the GNU Public License, the open source software license used in many open source applications, appears to have the ability to expose proprietary software developers to substantial risk if used improperly. This potential risk may act as a disincentive to using GPL'ed open source software in cumulative proprietary innovations, thus stunting innovation in general in the software market.

In closing, the net outcome of these effects on innovation in the software market is at best ambiguous. Nevertheless, as demonstrated in the market share data in section 3, open source software is a serious force that is shifting the traditional economic paradigm of the software market and is concurrently going to have to become a consideration for proprietary software firms in the formation of their economic strategies.

.

Bibliography

- Acs, Zoltan and Audretsch, David. 1987. Innovation, Market Structure, and Firm Size. The Review of Economics and Statistics, 69.

- Bonaccorsi, Andrea and Rossi, Cristina. 2003. Why Open Source software can succeed. Research Policy, 32.

- Fink, Martin. 2003. The Business and Economics of Linux and Open Source. Prentice Hall PTR, Upper Saddle River, NJ.

- Gomulkiewicz, Robert. 2002. De-bugging Open Source Software Licensing, 46 U. Pittsburg Law Review 75

- Gomulkiewicz, Robert. 1999. How Copyleft Uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B, 36 Houston Law Review 179

- von Hippel, Eric and von Krogh, Georg. 2003. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. Organizational Science, 14.

- Lerner, J., Tirole, J. 2002. Some Simple Economics of Open Source. Journal of Industrial Economics, 52.

- Mustonen, Mikko. 2003. Copyleft – the economics of Linux and other open source software. Information Economics and Policy, 15.

- Perens, Bruce. "The open-source patent conundrum." News.com. 31 Jan. 2005.

- Raymond, Eric. 1999. The Cathedral and the Bazaar. O'Reilly & Associates, Sebastpol, CA.

- Scotchmer, Suzanne. 1991. Standing on the Shoulders of Giants: Cumulative Research and the Patent Law. Journal of Economic Perspectives, 5.

- Scotchmer, Suzanne. 2004. Innovation and Incentives. MIT Press, Cambridge, MA.

- Scherer, F.M., 1980. Industrial market structure and economic performance. Rand McNally College Publishing Company, Chicago.

- Schumpeter, Joseph, 1942, Capitalism, Socialism and Democracy, New York: Harper.

- Schmidt, Klaus M. and Schnitzer, Monika, "Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market" (July 12, 2002). http://ssrn.com/abstract=319081

- Shankland, Stephen. “GPL gains clout in German legal case.” News.com. 22 Apr. 2004.

- Shankland, Stephen. “Microsoft spins ‘Halloween’ memos.” News.com. 6 Nov. 1998.

- Torvalds, Linus and Diamond, David., 1999. Just for Fun. HarperCollins, New York, NY.

- Tysver, Daniel. “The History of Software Patents.” BitLaw. 2000. 28 Feb. 2005 <http://www.bitlaw.com/software-patent/history.html>

- Weber, Steven. 2003. The Success of Open Source. Harvard University Press, Cambridge, MA.

- West, J. 2003. How open is open enough? Melding proprietary and open source platform strategies. Research Policy, 32.