Erasmus Law and Economics Review 1 (February 2004): 47–75.

TRADE SECRET VS. OPEN SOURCE: AND THE WINNER IS ...

Alberto Pianon*
Vicenza, Italy

Abstract

The contest between closed and open source concerns the trade secret protection of software source code and of the ideas embodied therein. Thus open source is not about giving away the "software machine" to users: it is about sharing its design and components amongst producers ("free as speech, not as a beer").

Trade secret has always been the main legal tool to protect ideas in software, given the latter's peculiar characteristics. However, the fact that open source software does exist and exhibits an impressive rate of growth in some fields, tells us that there must be a problem with trade secret, and that sharing ideas in software entails higher gains to developers.

This paper tries to explain the shortcomings of trade secret protection of source code (insufficient scope of protection, risks for data security and users' privacy, disincentives to developers, inefficient differentiation of products) and to find out what are the actual returns of open source developers from sharing source code (skills improvement and intellectual stimulation, reputation and career improvements, user-driven innovation).

However, the fact that not all software is open source suggests that sharing source code is not always rational, from the developers' point of view. Given that

* EMLE 2003, cum laude. The author practices as a lawyer in Vicenza, Italy. His office address is at P.tta Palladio 9/11, 36100 Vicenza, Italy. E-mail: alpianon@libero.it. This paper is partly based on his EMLE thesis submitted in August 2003. The author would like to thank to thank his thesis supervisor Prof. Vincenzo Denicolò for his availability and friendly support; Dennis Khong for the reading suggestions and the stimulating discussions via e-mail; Marc Pirrung for reviewing the article and providing numerous helpful comments. Any remaining errors are the sole responsibility of the author.

some costs related to trade secret protection mostly affect software users, and since open source seems to dominate mainly in fields where programmers are also lead users of the software they develop, one may argue that software producers choose open source only when they have to internalize all costs of inefficient legal protection. Therefore, open source may be in principle more socially efficient, but when the costs of trade secret are not fully internalized by developers, the latter tend to stick to the closed-source business model.

Keywords: Open source, Closed source, Trade secret.

JEL classification: K39, O34.

1. "Free As Speech, Not As a Beer"

Open Source Software/Free Software (OSS/FS)[1] *is basically a peculiar licensing system aimed at protecting the free sharing of software source code amongst programmers, which has been a common practice since the very beginning of the Software era*. The OSS/FS movement was born to fight against privatization of software source code[2]: even if most OSS/FS is freely available on the Internet, the focus is on freedom of source code sharing, not on price ("free as speech, not as a beer"). Since many authors seem to miss the very meaning of the "free speech" issue, it is opportune to clarify it from the outset.

Source code consists of a series of English-like instructions for the computer embodied in a text file, which can be understood by humans, but not directly by the machine. Software source code has to be translated (compiled) in another language to be executed by the computer, the so-called "object code" (which is the code we execute when we launch a program). In turn, object code (a sequence of 0 and 1) is not understandable by humans, but it could be partly re-translated (decompiled) into source code.

Notwithstanding software is actually a set of instructions *to* a machine, for

[1] The two terms identify the same kind of software, but with different ideological connotations. While promoting basically the same thing (a licensing system aimed at protecting the free sharing of source code), Open Source Initiative (OSI – www.opensource.org) and Free Software Foundation (FSF – www.fsf.org) are very different in their vision: while the latter labels closed-source software as "unsocial, that is unethical, that is simply wrong" (Stallman 2001), and advocates the Free Software choice as a matter of ethical values (freedom), OSI claims that there is no particular ethical case against closed-source, but open source software proved to be more efficient, and for this reason (efficiency, not freedom) corporations should adopt it. However, this is not a mere ideological debate, since OSI "allows" more permissive licensing schemes than FSF does with respect to the commercial exploitation of derivative works.

[2] A brief but illuminating history of OSS/FS can be found in Lerner & Tirole (2000).

convenience I will use a metaphor taken from Samuelson et al. (1994), who define software *as* a machine that performs certain actions. Such "machine" is made of "components" which have to be built and assembled together in an engineering process, which resembles the engineering and assembling of complex mechanical devices.

According to this analogy, *object code is the "machine"*, as it is sold on the market; *source code embodies its components and design, as well as the ideas* which are the basis of its functioning. The decompiling of object code into source code may be compared to the reverse engineering of a machine.

Therefore, the issue at stake with OSS/FS is not to give away the software machine to consumers, but to share its ideas, design, and components with other producers.

My understanding is that software developers decide to share ideas amongst them because traditional legal means of protection are not effective with respect to software, due to its peculiar – and sometimes unique – technical and economic characteristics. Thus it may make more sense to share source code and get returns in different ways.

To clarify this point, I would like to stress that many economists and legal scholars, when dealing with OSS/FS, mention the costs of legal protection of software, but they all focus on the issue of enforcing software licenses against piracy (i.e., copyright protection of *object* code against non licensed users).

Yet the rationale behind OSS/FS licenses is not allowing the sharing of programs (object code) amongst software *users* – there are open source companies that actually *sell* their software, and there is a lot of closed source software that is given away for free on the Internet, even by Microsoft. Indeed, OSS/FS licenses are aimed at protecting the sharing of ideas (source code) amongst software *programmers*.

2. Trade Secret, a Forgotten Issue

The ongoing debate about software protection seems to focus mainly on piracy (which concerns copyright on object code) and on software patents, yet the real issue at stake with OSS/FS is, in my opinion, *trade secret*. *Closed* source means precisely this: the source code of a software program is protected by trade secret, while according to *open* source licenses the code *must* be public.

Trade secret is not property (independent innovators cannot be excluded from using an invention protected by trade secret), however it *has many advantages with respect to other means of legal protection of software* (patent and copyright).

a) Firstly, trade secret protection can be obtained *immediately and at a relatively low cost*[3], while filing for a patent is expensive and time consuming: the value of innovations falls very quickly in the software industry, thus while the patent is pending the innovation may well become outdated.

b) Secondly, trade secret has a *broader scope of protection*: almost all valuable or useful information can be protected by trade secret[4], while only novel and non-obvious innovations may be protected by patents. As Samuelson et al.(1994) point out, innovation is mostly incremental and cumulative in software: "programmers both contribute to and benefit from a cumulative innovation process. *While innovation in program design occasionally rises to the level of invention, most often it does not*. Rather, it is the product of the skilled use of know-how to solve industrial design tasks" (p.2331). *Therefore most innovations in software cannot be patented, lacking the non-obviousness and novelty requirements*[5].

c) Thirdly, *expected litigation costs are far lower than with patents*. The violation of trade secret consists essentially of a breach of a non-disclosure obligation, and the only problem is the proof of such breach (and damage quantification, if penalties have not been set ex ante). Instead patent lawsuits often require subtle legal and technical discussions about the validity and the scope of protection of a patent, which in turn imply higher costs of litigation and more uncertain outcomes, so that the mere threat of a patent suit may be used as an unfair means of getting negotiating power.

d) Fourthly, *since innovation in software is mainly cumulative*[6], *anti-commons*

---

[3] In the software case, it is sufficient: *a)* to distribute the software in object code form, so that the "machine's" design and components are not immediately visible from "outside"; *b)* to sign non-disclosure agreements with employees and collaborating professionals and corporations; *c)* since misappropriation by the latter subjects can hardly be detected ex post, it is also necessary to enact security measures in order to prevent misbehavior: protected working areas (thus employees may not work at home and generally export source code from the internal electronic repository without authorization), compulsory security measures for collaborating firms, and so forth.

[4] It is sufficient that it is confidential and not generally known (generally, it is enough to prove the existence of confidentiality agreements and of reasonable security measures to protect it).

[5] Novelty means that the invention is different from all previous inventions ("prior art"). Non-obviousness implies that someone who is skilled in the particular field of the invention would view it as an unexpected or surprising development.

[6] Software programs, as physical machines, are inherently compilations of sub-components. Indeed, the distinction between a component and a sub-component is relative, since the prefix indicates only the hierarchy between them. In other words, programs are built from programs, in an industrial design process which assembles together different parts that may come from diverse sources. This industrial design process relies upon a large body of

issues may (and, in fact, do) arise with patents[7]. It has to be pointed out that anti-commons problems do not come only from hold-up strategies of patent holders; there are also the transaction costs of finding all patents that a new project may infringe upon, since texts describing patents are easily available on the Internet, but they are written in a complicated legal language, and often even the programmer does not recognize her/his own invention in such texts.

Furthermore, anti-commons problems are particularly severe because the software market exhibits a high degree of specific investments by users and by producers of complements[8], as well as relevant network and market externalities[9].

Summing up, software patents may grant a disproportionate and unjustified market power to the patent holder. This is the reason why huge software corporations share their knowledge by cross-licensing or patent pooling practices (i.e., they barter and share their patent portfolios). However, this kind of knowledge sharing is strictly limited to a few companies, leaving out small firms with limited patent portfolios (Economist Jun.23,2001). Therefore, patent rights are used as barriers to entry by incumbent firms.

For such reasons, many law and economics scholars suggest that the threshold of protection for software patent should be set very high, and that compulsory licensing

---

skilled know-how, acquired by software developers both from formal training and practice (Samuelson et al. 1994 p.2329).

[7] A programmer may need to bundle others' source code (or ideas) to develop a new software, but owners of patent rights may use hold-up strategies that could make it not worthwhile, from the programmer's standpoint, to develop the new software, even if it is socially efficient - as it always happens when something valuable is split among different property rights holders.

[8] Users spend much time learning how to use a software, and create files that are specific to that software and which often cannot be used with other programs (i.e., they cannot be – easily – shared with users of different softwares). Internet sites developers need to optimize the code of their pages in order to have them displayed correctly on a specific browser. Software applications producers have to write their software so that it can interface with a particular operating system. Developers need to program operating systems to work on specific hardware platforms - and the examples may go on and on. Such specific investments make users and producers of complements absolutely dependent on the software they choose in the beginning.

[9] Software programs, in order to perform their function, must interact with hardware components, with other software programs and with final users; therefore, there is a relevant degree of complementarity between different products, *i.e. market externalities* are widely present.
Furthermore, because of specific investments (see note 8), *network externalities* play a very important role in software markets. Because of specific investments, the utility of a software user – as well as the profit of producers of complements to that software – are affected by the number of other users purchasing the same software.

(i.e., transforming patent law from a property rule to a liability rule) may be desirable in some software-related cases (see Menell (1999), p.141-2, for a complete overview of the literature).

Yet, *trade secret, being not property, does not prevent independent innovators from using the same idea.* Therefore also in this respect (anti-commons issues) trade secret is more efficient than patent in protecting software innovation.

e) Finally, as to copyright, it should be noted that in principle software, as 'functional information' (*i.e.*, information that makes hardware perform functions), does not fit into the traditional requirements for copyright protection. Copyright is thought to protect "expression", as opposed to "ideas". However, neither the text of source code nor the object code "express" anything: software processes information, i.e., it performs functions. Copyright is not intended for protection of "functional" features, the "functionality" criterion being the dividing line between copyright's and patent's scope of protection.

Yet, since both source code and object code are easily duplicable at zero cost, Copyright Law is well suited to protect software producers against this very cheap way of free riding on their work[10] (Karjala 1998).

However, copyright is – at least in theory – a sufficient legal tool to protect software *object code* against piracy[11], but its scope of protection is totally inadequate to protect the design and ideas embodied in *source code.*

As Samuelson et al. (1994) clearly state, "at virtually all levels of abstraction above the literal text, a program consists of procedures, processes, systems, methods of operation, or their components" (p.2353): all features that cannot be protected by copyright. If this were the case, copyright would offer an unjustifiably long and broad protection to ideas, far beyond what is socially desirable. Consequently copyright may protect no more than the mere literal copying of source code.

Summing up, it should be no surprise that *trade secret (i.e., "closing" source code) has always been the main legal tool adopted by software houses to protect innovations and ideas against other software producers*, and that patents and copyright play only a secondary role in that respect, because of the problems that

[10] Indeed, in the very beginning copyright was introduced precisely to prevent unauthorized copying of texts when printing was invented and copying costs decreased dramatically (Khong 2003)

[11] New technologies make it increasingly easier to copy information, thus the problem is the *enforcement* of copyright in the digital age, not its legal scope of protection. I will not treat this issue here; a thorough analysis can be found in Menell (2002).

have been pointed out above[12].

3. What is the Matter with Trade Secret?

Even though there is a strong case for trade secret protection of source code, the OSS/FS movement is born to fight against it and to defend open access to innovations developed by others. There are three possible explanations (yet the first two are implausible, from a Law and Economics point of view).

The first explanation is altruism: yet as Lerner and Tirole (2000) cleverly note, "any explanation based on altruism only goes too far. While users in less developed countries undoubtedly benefit from access to free software, many beneficiaries are well-to-do individuals or Fortune 500 companies. Furthermore, altruism has not played a major role in other industries, so it would have to be explained why individuals in the software industry are more altruistic than others" (p.2). The same holds for explanations based only on political or cultural motivational factors.

The second explanation may be that open source programmers, for some reason, exhibit irrational behavior. However, this should imply that even large corporations such as IBM or Hewlett-Packard, which decided to invest huge sums in OSS/FS development, are acting irrationally, too.

In this regard, it should be highlighted that the idea, still rooted in some people's mind, that OSS/FS developers are just a band of teenagers programming at home in their spare time, is completely wrong. A recent survey of a sample of projects hosted by Sourceforge.net[13] has shown that OSS/FS programmers are mostly experienced IT professionals, about 30 years old on average, who collaborate through the Internet across corporations they work for in a very flexible and informal way, and who are globally distributed, mostly in North America and in the European Union. Furthermore, programmers who substantially contribute to open source projects are either allowed or even paid by their employers to do it during working hours (Boston Consulting Group 2002) This survey substantially confirms the results of another research project concerning only the Linux kernel developers (Hertel et al 2000-2003).

Therefore, given that OSS/FS programmers cannot be altruists or irrational

[12] To be honest, it should be highlighted that in most industries, except pharmaceuticals and chemicals, patents are not the principal means of appropriating returns from ideas, and trade secret plays the major role in this direction (for a complete report of empirical evidence, see Menell (1999) , p.136 and p.152)

[13] Sourceforge.net is the largest repository of Open Source code and applications available on the Internet, that provides free services to Open Source developers.

individuals (nor can be their employers), the only plausible explanation – that I will discuss below – is that there must be a problem with trade secret protection in software, and that it may be possible, for software programmers, to get utility returns by diverse sources, some of which have not been adequately considered, in my opinion, by economists and legal scholars.

In the next sub-sections I will treat the shortcomings of trade secret protection in software, that may be bundled in three categories: insufficient scope of protection, disincentives to developers, incentives towards inefficient differentiation of software products.

3.1 Insufficient Scope of Protection

As we have already reminded, trade secret is not property, i.e., it does not protect the inventor against third parties' independent discovery or legal acquisition of the information (i.e., by accidental revealing or reverse engineering).

Therefore trade secret cannot protect near-the-face[14] innovation, and unfortunately such a kind of innovation is crucial in software.

Computers have to interact with human beings, therefore their "behavior"[15] with respect to the final user turns out to be fundamental: computer-human interaction not only makes human tasks easier, but may also change radically the tasks themselves, allowing one to do things that in the real world are simply impossible.

Indeed what software users buy is not source code or object code (they do not become owners of the software): what they buy – by means of user licenses – is the "behavior" of the software "machine". In some cases *the most relevant research and development costs of a software house are related to "behavior" design* (i.e. the design of the functions that the software will perform to satisfy users' needs).

Unfortunately, "behavior" cannot be protected by patents because it is mostly a result, not a means to achieve a result. Yet "behavior" cannot be protected by trade secret either, since it consists of near-the-face innovation. Therefore competitors cannot be prevented from copying "behavior", free-riding on the innovator's investments.

But protecting trade secret is a problem even as to non-near-the-face inventions,: complex software projects may involve hundreds or even thousands of developers,

[14] Near-the-face innovations are innovations that cannot be kept secret and marketed at the same time, since they are visible from outside.

[15] I always follow the metaphor suggested by Samuelson et al.(1994) .

and avoiding any leak of source code (by insiders and/or outsiders) may be technically impossible.

Nor may copyright help in this respect, since source code which is protected by trade secret cannot be registered at the Copyright Office (otherwise it would be made public[16]) and cannot be checked by outsiders: therefore it may be difficult both to detect and to prove copyright violations on closed source code.

The legal impossibility to *appropriate* source code via trade secret, and the technical difficulty to keep it secret, induced software corporations to try to force first copyright[17] and then patent[18] beyond their "natural" boundaries, leading to an overprotection risk: in fact if copyright were extended to ideas, it would grant an unjustified broad protection; if the patent threshold of protection were lowered, anti-commons issues may paralyze technological development in software (or increase the barriers to entry for firms with small IP portfolios to offer and share with incumbent companies).

3.2 Security Issues

Trade secret entails risks as to data security and users' privacy. In fact software often processes confidential information, and backdoors or other tricks may be voluntarily or involuntarily inserted in the source code by the firm (or by a bad or distracted employee), allowing the possibility for whoever knows them to get access to the unaware users' private information. The power of control on information is a major issue, since even if it entails illegal conduct, *when the code is closed (i.e. secret) such conduct is very hard to detect.*

One notable example is the Borland Interbase case: after Borland decided in 2001 to open the source of Interbase (a widely-spread professional database software), it was brought to light, by the scrutiny of the worldwide OSS/FS community, that some of Borland's developers in 1994 "involuntarily" forgot an "innocent" backdoor which allowed access to all databases managed with Interbase. Borland recognized at the time that such a security hole was a very severe problem. If Interbase had not been transformed in an Open Source project, probably the backdoor

[16] US Copyright registration rules for software programs (http://www.copyright.gov/circs/circ61.html) provide that source code containing trade secrets can be registered by depositing a printout of the code with "trade secret portions blocked out" (even if such portions "must be proportionately less than the material remaining"). Obviously for such portions there is no legal evidence as to the content of the creation.

[17] See Menell (2002) .

[18] See Samuelson et al.(1994)

would have never been discovered[19].

This appears to be one of the main reasons why governments required Microsoft to disclose (part of) Windows source code to selected entities and agencies[20].

3.3 Disincentives to Developers

The difficulties of appropriating returns accruing from source code and security issues are not the only problems with trade secret.

Indeed, the implementation of trade secret deprives developers of most of the value they produce: part of this value is appropriated by IP lawyers and by the corporations they work for, part is simply dispersed or destroyed.

a) Authorship (reputation) is de facto taken by corporations

By and large, when a developer works as an employee for a software corporation, the Law provides that the intellectual property on the code she writes (copyright and patents) is automatically assigned to the corporation. Generally, authorship and reputation for the work done remains with the developer[21], at least in theory.

Yet, if the source code is covered by *trade secret*, nobody from outside the corporation can look at “who did what when” in the software.

Furthermore, software corporations are afraid to publish the names of their best programmers, since they fear that other corporations could head-hunt them by offering better job opportunities (Lerner & Tirole 2000).

Since the right to be recognized for one's own intellectual work appears to be somehow inscribed in human nature, programmers working for such corporations often find hidden ways to reveal their names, by inserting so-called “Easter Eggs” inside programs[22].

---

[19] Source: Shankland (2001), ZDNet News

[20] Source: BBC News Jan.15,2003

[21] According to Italian Law, authorship is inalienable, while in some countries one may even sell it.

[22] For instance, in Microsoft Word 97, one opens a new document, type "Blue" (without quotes), select the word Blue, go to Format -> Font, choose Font Style Bold, Color Blue, type " " (space) after word "Blue", go to Help -> About, keep Ctrl-Shift pressed together and at the same time click with the mouse on the Word Icon on the left of the banner. A screen with a pinball and all the names of Word developers will appear. Similar tricks are present in other programs (if one searches “easter eggs” on www.google.com, one can find lots of sites documenting this).

Instead, the reputation for the intellectual work in closed source software is in fact gained by the corporation; everybody associates Word with Microsoft, not with the individuals who actually wrote the program.

It should be noted that this is a typical problem with *trade secret*: in fact in patents the inventor's name is rendered public together with the invention, even if they work as employees – their firm would result as the assignee of the patent, not as the inventor.

However, one may argue that in small software houses, where the developers are also the owners of the firm, such problems would not occur. But this is only partially true: people will directly associate the software program to the developers' reputation, but since source code is protected by *trade secret*, the effective value of their work can be only partly appreciated from outside.

b) Adverse election

It has been widely demonstrated that productivity of software developers may vary radically from person to person, and on average about one in five programmers has a negative productivity, i.e. she slows down others' work, reducing the quantity/quality of the final outcome[23] (Feller & Fitzgerald 2002). Therefore attracting skilled developers and training novices are key issues in the software industry. Adverse selection problems are crucial, because for software developers more than other workers the proof of the pudding is in the eating: the most relevant part of a developer's skills reveal themselves only by observing her at work.

However, in the closed-source business model it is impossible to look at one's previous work, since source code is protected by trade secret. Due to the fact that

[23] This problem is somehow related to the so-called "software crisis": in a well-known – at least amongst programmers – article, W.W. Gibbs points out that, while hardware technology improves at an incredible rate of growth, the software industry seems unable to keep pace with it: "Studies have shown that for every six new large-scale software systems that are put into operation, two others are canceled. The average software development project overshoots its schedule by half; larger projects generally do worse. And some three quarters of all large systems are "operating failures" that either do not function as intended or are not used at all" (Gibbs 1994).
While the software crisis may arise from different causes (the treatment of which goes beyond the scope of this paper), a crucial point has been clearly highlighted by Brook's Law: "Adding manpower to a late software project makes it later" (Brooks 1995), as a result of the fact that the expected advantage from splitting work among n programmers is *EA(n)*, but the complexity and communications cost associated with coordinating and then merging their work is $EC(n^2)$. *I.e.*, while the overall productivity increases proportionally with the number of developers, the cost of coordination increases quadratically. Brooks therefore disproved the myth that "programmer time is fungible", and that the "Man-Month" could be a valid unit of measure for software production process.

potential employers cannot look at a developer's previous work, an adverse selection issue arises as to the recruiting of good developers. The possibility of knowing the actual skills of a developer is important in order to decide whether or not to employ her. If this is not possible, mostly mediocre programmers will be hired, and good programmers will try to find a job in another way, since the average wage is too low to reward their skills (to some extent, this *is* happening in reality, since OSS/FS professionals earn higher salaries[24]).

c) Closed-Source business model is not a good environment for creative work

The necessity of protecting trade secret has many consequences also on the organization of work: due to the ease of copying source code and "exporting" it from a firm without leaving obvious clues, major security measures have to be enacted, affecting the way software developers do their job.

When in 1981 Microsoft won the contract with IBM to develop an operating system for personal computers (DOS, Dirty Operating System), IBM compelled them to work in a small room without aeration, protected by locks and barbed wire, to prevent possible leaks of information. It is said that in August, when the inside temperature climbed up to 38°C, a Microsoft developer opened the door to air the room, and IBM's security personnel immediately reported this "severe" infringement of security measures (Staglianò 2000).

This is of course an extreme example, but it may give one an idea of how the practical implementation of trade secret may affect the everyday work of software developers, who more than likely may want to enjoy the choice of working where and as they like. After all *programming is to most developers a creative activity, requiring freedom of work and self organization* (see Boston Consulting Group Survey 2002).

Furthermore, always in order to better protect trade secret, it is common practice in huge software houses that different divisions work completely separate from each other, without the possibility of knowing anything about the others' work. Given the inherent difficulties of producing reliable and bug-free software programs this practice may be not only frustrating for developers but also counterproductive for the organization of work, fostering the so-called "software crisis"[25].

3.4 Incentives Towards Inefficient Differentiation of Products.

Another problem with trade secret is that it provides incentives towards inefficient

[24] See note 31
[25] See note 23

differentiation of software products. In this regard, for the sake of exposition I will take some steps backwards to clarify the issue of competition within software markets.

As we have already pointed out, *market and network externalities*, as well as *specific investments* play a very important role in software markets.

Because of specific investments, the utility of a software user – as well as the profit of producers of complements to that software – *are* affected by the number of other users purchasing the same software. As a consequence, the number of users of a particular software also affects the quantity and variety of complements available to that product (Shy 1995).

The higher the value of specific investments related to software, the larger the probability that users and producers of complements are better off if a market niche is dominated by a single *de facto* standard software, rather than in a situation with strong competition among fungible but incompatible products. More precisely, if software quality and performance are homogeneous, and the diversity lies only in the interfaces, strong monopolistic competition may discourage specific investments and reduce dramatically users' and complement producers' returns from such investments, compared to a market dominated by a *de facto* standard software. Moreover, the possibility of multiple equilibria in competing standards markets and the consequent uncertainty (about whether only one standard will prevail and, in such cases, which one will prevail) may discourage or delay specific investments, especially from users or complement producers with a high discount rate.

With respect to what has just been said, one may note that in the real world software programs do differ in performance and quality. Therefore one may argue that there could be a trade off between the exigence of early standardization in order to encourage specific investments, and allowing leeway in the experimentation of different software solutions, in order to improve performance and quality.

But as we have already noticed, innovation is mostly incremental in software – i.e. it is built on existing software programs. Therefore investments in innovation are mostly *specific* to the particular software(s) which they are built upon. Consequently, the trade off between early standardization and allowing leeway in experimentation exists only for non-incremental innovations, i.e., for true inventions (David & Greenstein 1990).

Furthermore, as for incremental innovation, it should be noted that if there are many firms producing incompatible (but similar) software products, each of them may invest in incremental innovation only on its own product, and not on others'. Suppose that there are 30 firms producing 30 incompatible versions of the same

software, or 30 software programs which perform the same tasks, and that each firm invests 100 in its own product's incremental development: if they merged into only one firm, there would be no economic rationale to develop 30 incompatible programs: only one product would be produced, and the investment on that product would be 3000.

We have pointed out that complex software production entails a specific problem (the so-called "software crisis"[26]): this means that incremental improvements are continuously required even after a program is launched on the market, since it is quite impossible to develop a complex software that works smoothly before testing it in all the unforeseeable situations of everyday use – i.e., the most relevant part of bugs are usually discovered only after the software is marketed.

Consequently, the quality of a software depends mostly on specific investments on incremental innovation, both before and after its launch on the market.

Thus early standardization is desirable in order to encourage (and to avoid dispersion of) investments in incremental innovation and to improve software quality.

So when is software diversity desirable, from an economic standpoint?

Software diversity in monopolistic competition, may arise from three different elements:

1) the producer may provide significant new features, which constitute a real – patentable – invention with respect to the state of the art;

2) the producer may provide some incremental innovations with respect to the other firms;

3) the producer may develop and implement specific interfaces (incompatible with other programs) in order to lock in users and producers of complements. It should be noted that the latter is often the easiest, cheapest and most effective way to achieve the goal of differentiating one's own software product, even if there is no very innovation.

It should be clear now that software diversity in the same market niche may be desirable only in case 1) – i.e., *if, and only if, there is (still) room for non-incremental innovations.*

Unfortunately, competing firms have an incentive to develop diverse products also in cases 2) and 3), especially if such a diversity may be protected by trade secret.

[26] See note 23

Therefore, in the closed-source business model, market concentration is in my opinion the only way to obtain early standardization, since incentives to inefficiently differentiate would disappear. However, market concentration entails monopoly costs, because the dominant firm may abuse its market power.

In other terms, *the business model based on trade secret leaves us with the choice between bearing the costs of inefficient software diversity*[27] *and bearing the costs of monitoring a dominant firm to prevent or punish possible abuses*[28].

4. Who Bears the Costs of Trade Secret?

Summing up, trade secret *a)* has an insufficient scope of protection as to software, *b)* may entail severe risks as to data security and users' privacy, *c)* provides disincentives to developers and *d)* fosters inefficient differentiation of products in a market that instead requires – in most cases – early standardization.

One should add that the most common complaint amongst software developers is that by and large the returns of trade secret protection do not go to them, since a lot of intermediaries (software corporations which they work for, law firms) eat up most of the revenues accruing from the legal protection of programmers' ideas.

More generally, all instruments aimed at protecting and "fencing" ideas (patent, trade secret) are artificial constructions of the legal system, which entail high administration and litigation costs, and – like in the software case – may not be very effective. Therefore, if the gains from protecting ideas are lower than the gains from sharing them, a rational individual will choose the latter solution.

The fact that open source software does exist, suggests that there must be some returns accruing to developers from source code sharing, which I will try to list and discuss in the next section.

The fact that not all software is open source, may suggest that there are some variable elements which in some instances make it more efficient (from developers' point of view) to protect source code by trade secret.

In this regard, as to the shortcomings of trade secret protection we mentioned before, it should be pointed out that

- *a* (insufficient scope of protection) and *c* (disincentives to developers) affect software programmers;

[27] Such as in the Unix case: it seems that there are about 30 different and incompatible versions of Unix operating systems in the market (source: Young 1999)

[28] Such as in the Microsoft case.

- *d* (inefficient differentiation) concerns partly programmers, partly software users (both make specific investments);

- *b* (security risks) affects only users (who often are not fully aware of such risks, if they are not computer experts).

Indeed, open source software seems to dominate in software markets where software producers and software users (partly) coincide[29] so that the costs related to *d* and *b* are internalized by producers (and the information asymmetry concerning *b* disappears).

This may suggest that, from a social efficiency standpoint, open source could be superior, but when the costs related to trade secret protection are not fully internalized by producers, the latter decide to keep the source code closed. I will discuss this issue later on, in section 6.

5. What are the Returns from Sharing Source Code?

The economic models which try to explain what are the returns from sharing source code, in my opinion can be grouped into three categories: community models, reputation-based models, user-driven innovation models. However such models tell us only a part of the story. Therefore I will add a further explanation, based on recent surveys about open source programmers' actual motivations, namely the Boston Consulting Group Hacker Survey 2002 and the “Free/Libre and Open Source Software (FLOSS) Survey and Study” carried out by the International Institute of Infonomics at the University of Maastricht (Gosh et al. 2002).

5.1 Community Models

The first group of theories argue that software developers contribute to OSS/FS projects because they can reasonably expect that many other programmers will contribute to the OSS/FS project by fixing bugs, adding features and so forth. Community models explain such cooperation in many different ways, from altruistic behavior to game theory.

---

[29] Namely, applications and operating systems for high-end users (who are also programmers).

The Apache and, in its early stage, the Linux case, are clear examples of that: Apache has been created by a pool of webmasters who needed a reliable web server program to work on; Linux has been initially created by Linus Torvalds and his first contributors with the purpose of having a Unix-like operating system working on their – at the time, new – intel 386 platforms (see Lerner & Tirole 2000; Torvalds 1999).

At the time of this writing, Apache has a market share close to 70%, always growing (source: http://news.netcraft.com/archives/web_server_survey.html). This means that about 70% of all existing Internet sites run on Apache.

Bessen (2001) argues that OSS/FS project initiators choose to give away their code to the OSS/FS community because it is more efficient than closed source companies in debugging complex projects. Bessen assumes that the cost of developing a software with M features is $(M+1) \cdot C$, where C is the cost of coding a single feature, but the cost of debugging suffers from the fact that every combination of features requires to be tested and debugged separately (that is what happens mostly in reality). Therefore the cost of debugging rises exponentially with the number of features ($2^M \cdot D$, where D is the cost of debugging a single combination of features). In other words, the cost of thoroughly debugging a software is prohibitive for any closed source company (because of the so-called "software crisis"), therefore if a programmer wants to save debugging costs and at the same time have the scrutiny of a worldwide developers community (which may be able to do the job better than a firm), she/he will decide to open the source code and wait for contributions.

However, while this model may explain why OSS/FS project initiators decide to give away the code to the community, it still leaves us with no explanation as to why the community members decide to collaborate for free.

Johnson (2001) builds a game-theoretical model in order to analyze the decision of a programmer to develop a piece of software as OSS/FS, according to her/his rational expectation about the decision of other programmers. When a software project has many tasks to be completed, and there is a relatively high number of potential developers, the game will end in an equilibrium where programmers decide to contribute. As Johnson notes, "good open source projects need to be developed initially by a small group and only later released to the general community for further improvement. Heuristically, developers need to have something sizable to 'play with' before the open source model can be expected to do well" (p.24).

While such theories may explain the advantage of OSS/FS in managing complex projects, they do not take into account that the vast majority of OSS/FS projects are developed by a few individuals, without any community help[30]. Former explanations suggest that it would not be economically convenient to develop small projects as

[30] A recent survey on SourceForge.net (Krishnamurthy 2002, as reported by Waterman 2003) found that "the vast majority of OSS programs are developed by only a few individuals. *The average number of developers was 6.61, the median was 4, and the mode was 1. Only 30% of projects sampled had more than 5 contributing developers, and only 19% of the projects sampled had more than 10 developers. Many projects did not tend to benefit from input from the rest of the community too:* the average project sampled had only two forums and two mailing lists for discussion, 10% of the projects had neither a forum nor a mailing list, and no messages were ever posted to the forums of 33% of the projects". It should be underlined that this survey analyzes only *mature* (i.e. successfully completed) projects, therefore it cannot be claimed that the small average size of OSS/FS projects is due to their early stage of development.

OSS/FS: but since reality shows the exact opposite, a further economic rationale has to be found in order to solve the problem of OSS/FS developers' motivation.

5.2 Reputation/Signaling Models

In the reputation/signaling models, programmers develop OSS/FS software in order to gain reputation among their peers – to get ego gratification and esteem – and/or outside the OSS community – "to be rewarded with private-sector job offers, shares in commercial OSS-related companies or future access to venture capital markets" (Waterman 2003). This is possible because in OSS/FS the individual work can be observed by anyone, contrary to what happens in Closed-Source Software.

Lerner and Tirole, in their seminal article *The Simple Economics of Open Source* (Lerner & Tirole 2000), point out that the core motivational factor for OSS/FS developers is career concern incentives: programmers receive delayed payoffs in terms of better job opportunities, higher contractual power to negotiate wages or to get access to financing if they want to start their own enterprise.

Starting from these assumptions, some authors (Leppämäki & Mustonen 2003, Lee et al. 2003) built models in which developing OSS/FS is used by good programmers to signal their innate skills that otherwise would be hardly noticed by potential employers or by financing institutions. In most cases such a signal is too costly to be duplicated by bad programmers. However in Leppämäki & Mustonen's model, in the case of strong positive market externalities (complementarity between OSS/FS and commercial software), the game may inevitably end in a pooling equilibrium.

In my opinion the reputation/signaling models at the moment constitute the most powerful economic interpretation of OSS/FS, and they seem to fit empirical data about the higher wages of OSS/FS professionals compared to Closed-Source professionals[31].

Yet the problem is that observing the reality one can notice that there are other elements which do not perfectly fit such theories.

---

[31] Open Source professionals' average salary is substantially higher than Windows Certified Professionals', due to an increasing demand from the IT labor market. In November 2000 the Linux professional's average salary was $82,500 per year ($2,000 to $3,000 more than the year before), and the average contract rate was $80 per hour. Apache or Sendmail professionals earned similar figures (Source: IT Salary Tracker – Datamation/EarthWeb's Dice job service). On the other hand, *Certified* Microsoft professionals for Windows 2000 in 2000/2001 earned on average $67,100 per year, while non certified professionals made on average $63,000 per year (Source: Microsoft Certified Professional Magazine) - that means Linux professionals earn about 1/3 more than Windows experts.

Firstly, there is much more cooperation (i.e., less strategic behavior) than predicted by these economic models: for instance, as McGowan (2002) noted, "because open-source licenses allow forking[32], these assumptions imply that such programmers will engage in strategic forking to enhance their reputational returns. If programmers in general care only about the size of the project they work on – and thus the extent of their reputational gain – one would expect programmers to accept strategic forking and work on the project run by the maintainer who most credibly promised to maximize programmers' reputations on her project. But I have seen no reports of cases in which forking occurred simply to enhance reputation" (p.39-40). In fact, as Raymond (2000) highlights, "there is strong social pressure against forking projects. It does not happen except under plea of dire necessity, with much public self-justification, and with a renaming".

Secondly, if OSS/FS developers were motivated only by reputation, mundane work (such as writing documentation, developing easy-to-use interfaces for inexpert users, updating drivers for out-of-date hardware and the like) would not be done at all (McGowan 2002). However, as Linus Torvalds claims, "the mundane work has to be done as well, of course, but people *do* do it. Sometimes it happens just because they are paid to do it, but quite honestly, more often (I think) because the people involved are just proud of what they do, and dotting the i's and crossing the t's is part of making a good product" (Torvalds, Interview July 8, 2003).

Thirdly, and most importantly, recent surveys, based on interviews of significant samples of OSS/FS developers, showed that the main benefits, which developers effectively receive by participating in OSS/FS projects, are not reputation gains or career opportunities.

According to the Boston Consulting Group Hacker Survey 2002, the main benefit of participation in OSS/FS projects is, for 48.1% of interviewed developers, "personal knowledge improvement", and for 26.3%, "personal sense of accomplishment" for the work done. The benefits related to reputation and career account for only 11.5% of total answers.

The Boston Consulting Group Hacker Survey 2002 also reports that the main motivators for OSS/FS developers are basically two:

a) often developers of an OSS/FS project are also its (lead) users; in other words, they are innovation "users" rather than innovation "manufacturers" (Von Hippel 2002).

---

[32] Open Source licenses allow anyone to take any OSS/FS product, develop it in different evolutionary directions at will, and claim that *that* is the product. This divergence is called "project forking" or simply "fork".

b) participation in OSS/FS projects is intellectually stimulating and improves programmers' skills and know-how;

Such results are also confirmed by other significant surveys, as the "Free/Libre and Open Source Software (FLOSS) Survey and Study" carried out by the International Institute of Infonomics at the University of Maastricht (Gosh et al. 2002).

As to the first point, it perfectly fits the user-driven innovation models, which we will deal with in section 5.3.

As to the second point, it should be stressed that programming skills are not innate, as signaling models assume (at most one may have an innate ease of learning). Indeed programmers improve their skills partly by formal training, and mostly "by developing programs themselves and by working with others in teams" (Samuelson et al. 1994, p. 2330): *practice in diverse software projects is crucial to form and improve a programmer's skills and know-how – and such practical know-how is the most valuable good to the developer and her/his employer.* We will try to explain why Open Source results in being more intellectually stimulating than Closed Source in section 5.4.

5.3 User-Driven Innovation Models

The third group of models concerning OSS/FS motivation stem from the fact that in many cases OSS/FS programmers are also users of the software they develop, i.e., not only are they producing innovation, but they also benefit directly from what they produce.

User-driven innovation is not a novel thing in industry, and in certain sectors it has proved crucial (Von Hippel 2002): in fact in some cases users may expect a higher reward from innovating than manufacturers (Von Hippel 1988), and in many instances innovation costs are lower for users than for manufacturers "when the problem-solving work of innovation developers requires access to 'sticky' - costly to transfer - information regarding user's needs and the context of use" (Von Hippel 2002 p.9). Moreover, user-driven innovation business models allow one to avoid wasting time in developing useless features and to focus only on users' needs.

The custom of giving away innovation is not a peculiarity of OSS/FS developers/users: there are many case studies in other industry sectors in which such practices turn out to be very common and widespread (sport equipment, scientific instruments and the like; but there are also examples in the chemical sector, as well as in other important industry fields - see *ibidem* p. 12).

The motivations, for users/innovators, for giving away their innovations, are related by and large to the presence of network and market externalities (innovators may exploit the fact that their free inventions become widely adopted because of network effects, therefore increasing the value of a complementary activity or asset managed/owned by innovators) and/or by the specificity of inventions to innovators, so that free riders would not gain an advantage equal to the innovator.

Such model explains also why in OSS/FS there is much more cooperation than predicted by reputation/signaling models. We have seen that inefficient diversification of software products imposes costs on users; yet when producers and users are the same people, producers will have an incentive to early standardize and to limit forking of software projects.

While user-driven innovation models provide another interesting rationale for OSS/FS, they cannot pretend to be a general explanation of motivation in OSS/FS: even if many OSS/FS projects are carried out by their lead users, there is no empirical evidence that *all* OSS/FS projects can be explained in terms of user innovation; yet occasional observation seems to suggest that there are many projects (or features of projects) which are developed regardless of their direct functionality to developers.

Therefore a further rationale has to be found in order to explain, from a general perspective, what is the core return from developing OSS/FS, regardless of its contingent (even if relevant) characteristics.

5.4 Skills Improvement and Intellectual Stimulation

As recent surveys show (Boston Consulting Group 2002, Gosh et al. 2002), the core motivational factor to OSS/FS developers is that open source is more intellectually stimulating than working in a traditional firm producing closed-source software, and that participating in an OSS/FS project improves skills and know-how, in a way which appears non-duplicable in the closed-source business model. In my opinion, this is due *a)* to the creative freedom one may enjoy in open source, and *b)* to the way software projects are managed (peer-leadership).

a) Creative Freedom

In my opinion OSS/FS offers such a freedom of action and a range of possibilities that are impossible to replicate within firms.

Firstly, the costs to start or to enter in an Open Source project (apart from writing the source code) are virtually zero: it is sufficient to write a piece of code, submit it via the Internet to the project maintainer, and if the latter accepts it, it is done; or, as to

the project leader, it is sufficient to post the project on, say, Sourceforge.net[33] (it is free), and to search for voluntary contributions; there is no need for employment contracts, non-disclosure agreements, protected working areas to defend trade secret, and so forth – as well as no need for lawyers, accountants, managers and the like to administer intellectual property rights.

The incredible rate of growth of the number of projects hosted by Sourceforge.net[34] is a clear indicator of the success of this model.

By and large, most developers may voluntarily choose (with the *permission* of the maintainers, who accept or reject the contributions) which project(s) to participate in, how much time to dedicate to each project, what features to develop and so forth; there are no legally binding agreements or deadlines – strict deadlines in software production do not make much sense, if one considers that such deadlines are almost never met and that incremental innovations are almost always needed after the launch of the product; indeed, *software can be regarded more as a "process" rather than a "product" in the traditional sense*.

Furthermore, developers may work where, when and as they like: there is no need to develop software in a specific place during specific hours since *there is no trade secret to protect* – i.e., there is no need to monitor and prevent programmers from "stealing" source code and selling it to competitors, because it would no longer be a theft. People can just cooperate freely through the Internet, having the possibility of working rapidly on fresh ideas without any legal limitation.

Such freedom gives developers incredible opportunities to participate in diverse and challenging projects without moving from their place and without the need of negotiating and signing contracts – opportunities they could never enjoy in the closed source business model, because the necessity of protecting source code puts up too many legal boundaries.

As recent studies show, software programming belongs to the category of creative jobs in which one can observe no clear separation between free time and work time, since working is perceived more as pleasure rather than as labor (Muffatto 2003). It is such a pleasure that 60% of OSS/FS developers declare that "with one more hour in the day, I would spend it programming" (Boston Consulting Group 2002).

---

[33] See note 13

[34] At the time of this writing (February 2004) the number of OSS/FS projects hosted by SourceForge.net – is about 75,500, and it grows daily at an impressive rate - in May 2002, there were about 39,000, as reported in Lerner & Tirole (2002).

In other words, developers get utility from writing source code, regardless of the money they make. Therefore, speaking in economic terms (and simplifying a bit), their maximization problem seems to consist in the maximization of the intrinsic utility they get from programming (non-monetary returns), under the constraints of earning enough money to live decently and having access to hardware for their programming (Raymond 2000).

b) Peer-leadership

Another crucial point in my opinion is the way OSS/FS projects are managed. First, it is more stimulating for a project founder to know that he will be the only holder of the right to manage that project, and that the corporation, which he works for, cannot take it.

However, if he manages the project in a way that the (small or big) community that grows around that project does not agree, such a community has the power to contest him, since *they are not bound by enforcing contracts to cooperate*; because the leader needs the community, he has to continuously share his views and treat other developers as peers, and authority has to be continuously earned.

The majority of the interviewed hackers in the Boston Consulting Group survey agree that the most important tasks of an OSS/FS leader are not to recruit contributors, determine and delegate tasks, manage timing or provide motivation to contributors (like for traditional project managers). Instead, according to the community members' views, an OSS/FS leader should provide a valid code base and a vision of the project, and his core task is to initiate dialogue with contributors and to integrate submissions. The majority of the interviewed hackers in the above-mentioned survey agree that "the next best thing to having good ideas is recognizing good ideas from others".

This paragon of a leader is very far from the leadership model which dominates within closed source firms; peer leadership is possible in OSS/FS because authority is not exercised to compel people to work (because contributions are voluntary) but in order to coordinate work and assure that the program is developed within the framework of a coherent design.

5.5 Reputation is the Synthesis of All OSS/FS Motivational Factors

We have seen that reputation does not appear to be a main motivator of OSS/FS developers. However, to some extent this is not precise: while the main motivational factor is undoubtedly related to skills improvement and to intellectual stimulation, there is always the constraint of earning enough in order to be free to program.

Reputation is without any doubt a useful reward programmers get from "owning" their projects or by significantly collaborating on others' projects: the value of reputation may be intrinsic – satisfaction, esteem – but also extrinsic – the possibility to participate in important open-source projects and *to get a job in academic institutions or business corporations* (Lerner & Tirole 2000) .

Indeed, reputation may well synthesize the manifold motivations of open source software developers: be it "the joy of programming", the want of esteem from others or the possibility to gain status in the academic or business world, in any case one needs to gain reputation both to join more important projects (and get more "joy" from the creative work itself) and to get recognition of his programming skills inside and outside the movement (with all the positive side-effects it entails – career, job opportunities and the like).

Therefore, even if reputation may not always be an end as such, it is a major means for OSS/FS developers to pursue their ultimate goals.

The fact that OSS/FS licenses and community norms are strongly directed towards the protection of authorship acknowledgment (Raymond 2000) may indicate that reputation, together with creative freedom and peer-leadership, is one of the most important values (even if not the only one) which the OSS/FS property rights system tends to protect, in contrast with the closed source business model, where authorship is *de facto* taken by the corporation.

5.6 Open Source Software Projects are a Valuable Resource to Developers

Summing up, we have seen that because of the creative freedom OSS/FS developers enjoy and of the way OSS/FS projects are managed (peer-leadership), *working in an OSS/FS project is more intellectually stimulating and improves developers' skills and know how*. This has both an *intrinsic* and an *extrinsic value*: the intrinsic value is the pleasure of the creative work as such; the extrinsic value is the increase in the value of human capital, i.e. of developers' productivity.

We have also seen that *working in an OSS/FS project makes one's performance more visible from the outside, therefore increasing one's reputation*: such reputation has both *intrinsic* value (satisfaction, esteem) but also extrinsic (the possibility to participate in important open-source projects and to get a job in academic institutions or business corporations).

Finally, *in many cases working in an OSS/FS project has a direct value for developers because they also are its lead users* (user-driven innovation).

In a few words, participating in an OSS/FS project is a resource (i.e., a source of

value) for developers, because it grants them some major returns that are completely lost (or appropriated by legal professionals and corporations) in the closed source business model. This is the reason why open source programmers do not fence ideas, but they do "fence" software projects. In other terms, while OSS/FS licenses protect an open-access regime on source code, each software project is ruled in fact by a commons regime[35], with a maintainer (mostly the project founder) who decides who has read-write access to the "official" repository of the *original* source code (and therefore can modify it). However, this issue deserves a separate and apposite treatment, going beyond the scope of this paper.

6. And the Winner is ...

Summing up and trying to draw some conclusions, we should recall again that the contest between closed and open source concerns the trade secret protection of software source code and of the ideas embodied therein. Thus open source is not about giving away the "software machine" to users: it is about sharing its design and components amongst producers ("free as speech, not as a beer").

Trade secret has always been the main legal tool to protect ideas in software, given the latter's peculiar characteristics. However, the fact that open source software does exist and exhibits an impressive rate of growth in some fields, tells us that there must be a problem with trade secret, and that sharing ideas in software entails higher gains to developers.

This paper tried to explain the shortcomings of trade secret protection of source code (insufficient scope of protection, risks for data security and users privacy, disincentives to developers, inefficient differentiation of products) and to find out what are the actual returns of open source developers from sharing source code (skills improvement and intellectual stimulation, reputation and career improvements, user-driven innovation).

However, the fact that not all software is open source suggests that sharing source code is not always rational, from the developers' point of view. Given that some costs related to trade secret protection mostly affect software users, and since open source seems to dominate mainly in fields where programmers are also lead users of the software they develop, one may argue that software producers choose

[35] The terms "open access" and "commons" are used here according to Ostrom's distinction (Ostrom 1999): "open access" is a property regime in which nobody has the right to exclude others from accessing the resource; in a "commons" regime the resource is owned by a community, whose members have the right to exclude non-members, and which has rules *i)* defining how the resource can be managed and *ii)* assigning different roles and powers to different members.

open source only when they have to internalize all costs of inefficient legal protection.

Therefore, open source may be in principle more socially efficient, but when the costs of trade secret are not fully internalized by developers, the latter tend to stick to the closed-source business model.

Furthermore, the attempts of spreading open source software in markets where users are not developers (typically, the so-called “desktop-market”) tend to exhibit the same problems as the closed-source business model (lock-in strategies by commercial distributors and consequent inefficient differentiation of products).

In the end the question, that this highly informal analysis suggests, is whether it may be possible to design legal tools to allow full internalization of externalities in software markets where producers are not at the same time users of the software, in order to foster the spreading of open source even in those markets.

If this were the case, also competition issues would be definitively solved because even if open source software becomes a *de facto* standard, no one gets market power from it. Instead the business model based on trade secret leaves us with the choice between bearing the costs of inefficient software diversity and bearing the costs of monitoring a dominant firm to prevent or punish possible abuses.

However, if legal tools to foster open source in such markets cannot be designed and enacted, in my opinion monopoly is something we have to live with also for the years to come.

Bibliography

BBC News Jan.15,2003. *Microsoft to reveal source code.* Available at http://news.bbc.co.uk/1/hi/business/2659857.stm

Bessen J. 2001. Open Source Software: Free Provision of Complex Public Goods. *Research on Innovation.* Available at http://www.researchoninnovation.org/opensrc.pdf

Boston Consulting Group Hacker Survey 2002. Available at http://www.osdn.com/bcg

Brooks F., 1987. No Silver Bullet: essence and accidents of software engineering, *IEEE Computer Magazine*, April 1987, p. 10-19

Brooks F. 1995. *The Mythical Man-Month*, Addison-Welsey

David P.A. and Greenstein S., 1990. The Economics of Compatibility Standards: an Introduction to Recent Research. *Econ. Innov. New. Techn*. Vol., 13-41

Economist Jun.23,2001, S40-42. *Patently Absurd.*

Feller J. and Fitzgerald B., 2002. *Understanding Open Source Software Development.* Addison-Welsey

Gibbs W. W.,1994. Trends in Computing. *Scientific American*, Sept. 1994, p.86. Available at http://cispom.boisestate.edu/cis310emaxson/softcris.htm

Ghosh R.A. & Prakash V.V., 2000. *The Orbiten Free Software Survey.* Available at http://orbiten.org/ofss/01.html

Ghosh R. A., Glott R., Krieger B. and Robles G., 2002. *Free/Libre and Open Source Software (FLOSS): Survey and Study - Final Report, Part IV: Survey of Developers,* International Institute of Infonomics, University of Maastricht. Available at http://www.infonomics.nl/FLOSS/report/index.htm

Hertel, G., Niedner, S. & Herrmann, S., 2000-2003. Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel, *Research Policy* (forthc.). Available at http://www.psychologie.uni-kiel.de/linux-study

Johnson J. P., 2001. *Economics of Open Source Software*. Available at http://opensource.mit.edu/papers/johnsonopensource.pdf

Karjala D. S., 1998. The Relative Roles of Patent and Copyright in the Protection of Computer Programs, *John Marshall Journal of Computer and Information Law*, Fall 1998, 41-74. Available at http://www.law.asu.edu/HomePages/Karjala/Articles/JohnMarshallComp&InfLaw1998.html

Khong D. W. K., 2003. *The First British Copyright Act of 1710 and Its Struggle Against Monopolies*, Preliminary draft. Available at www.esnie.org/pdf/st_2003/papers/10_Khong.pdf

Krishnamurthy S., 2002. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. *First Monday*, Vol. 7, No. 6 (June). Available at http://www.firstmonday.org/issues/issue7_6/krishnamurthy

Lee S., Moisa N., Weiss. M., 2003. *Open Source as a Signalling Device - An Economic Analysis*, Goethe University, Frankfurt am Main, Working Paper Series: Finance and Accounting 102. Available at http://ideas.repec.org/p/fra/franaf/102.html

Leppämäki M. and Mustonen M., 2003. *Spence Revisited - Signalling with Externality: The Case of Open Source Programming*, Department of Economics, University of Helsinki, Discussion Papers, n.558-2003. Available at http://ethesis.helsinki.fi/julkaisut/val/ kansa/disc/558/spencere.pdf

Lerner J. and Tirole J. 2000. *The Simple Economics of Open Source*, NBER Working Papers. Available at http://papers.nber.org/papers/w7600

Lerner J. & Tirole J., 2002. *The Scope of Open Source Licensing*, NBER Working Papers. Available at http://papers.nber.org/papers/W9363

McGowan D., 2002. *Legal Implications of Open-Souce Software*, SSRN. Available at http://papers.ssrn.com/paper.taf?abstract_id=243237

Menell P.S., 1999. *Intellectual Property: General Theories*, Encyclopedia of Law and Economics, 129-188.
Available at http://allserv.rug.ac.be/~gdegeest/1600book.pdf

Menell P.S., 2002. *Envisioning Copyright Law's Digital Future*, Boalt Working Papers in Public Law, Paper 5.
Available at http://repositories.cdlib.org/boaltwp/5

Muffatto M., 2003. Unpublished Book on the Economics of Open Source

Ostrom E., 1999. *Private and Common Property Rights*, Encyclopedia of Law & Economics, 332-379. Available at
http://allserv.rug.ac.be/~gdegeest/2000book.pdf

Raymond E. S., 2000 *Homesteading the Noosphere*, Available at
http://catb.org/~esr/writings/homesteading

Samuelson P., Davis R., Kapor M.D. and Reichman J.H., 1994. A Manifesto Concerning the Legal Protection of Computer Programs. *Columbia Law Review* 2308, (1994). Available at
http://wwwsecure.law.cornell.edu/commentary/intelpro/manifint.htm

Shankland S., 2001. *Borland InterBase backdoor detected*, ZDNet, Jan.11,2001. Available at http://zdnet.com.com/2100-11-527115.html?legacy=zdnn

Shy O., 1995. *Industrial Organization*, MIT Press, chapt. 10, 253-277

Staglianò R., 2000. *Bill Gates: una biografia non autorizzata*, Feltrinelli

Stallman R. M., 2001, *The GNU project.* Available at
http://www.fsf.org/gnu/thegnuproject.html

Torvalds L., 2003. *What, me worry?* Interview with Stephen Shankland of News.com. Available at http://news.com.com/2008-1082-1023765.html

Torvalds L., 1999. The Linux Edge. *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates. Available at http://www.oreilly.com/catalog/opensources/book/linus.html

Von Hippel E., 1988. *The Sources of Innovation*, New York, Oxford University Press

Von Hippel E. 2002. *Horizontal Innovations Network - By and For Users*, MIT Sloan School of Management. Available at http://web.mit.edu/evhippel/www/UserInnovNetworksMgtSci.pdf

Waterman A. 2003. *Motivations to Develop Software as Open Source*, SIEPR, Stanford University. Available at http://www.stanford.edu/~awaterma/OSS/Waterman-OSSMotivations.pdf

Young R., 1999. Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry, *Open Sources: Voices from the Open Source Revolution*, O'Reilly & Associates. Available at http://www.oreilly.com/catalog/opensources/book/young.html