Copyright or Copyleft? An Analysis of Property Regimes for Software Development

Paul B. de Laat

Faculty of Philosophy, University of Groningen, Oude Boteringestraat 52, 9712 GL Groningen, The Netherlands

Abstract

Two property regimes for software development may be distinguished. Within corporations, on the one hand, a Private Regime obtains which excludes all outsiders from access to a firm's software assets. It is shown how the protective instruments of secrecy and both copyright and patent have been strengthened considerably during the last two decades. On the other, a Public Regime among hackers may be distinguished, initiated by individuals, organizations or firms, in which source code is freely exchanged. It is argued that copyright is put to novel use here: claiming their rights, authors write `open source licenses' that allow public usage of the code, while at the same time regulating the inclusion of users. A `regulated commons' is created. The analysis focuses successively on the most important open source licenses to emerge, the problem of possible incompatibility between them (especially as far as the dominant General Public License is concerned), and the fragmentation into several user communities that may result.

Keywords: commons, intellectual property rights, licensing, open source, software

* Tel.: +31-594-540155;
fax: +31-50-3636160.

E-mail address: P.B.de.Laat@philos.rug.nl.

Copyright or Copyleft? An Analysis of Property Regimes for Software Development

1. Introduction

In the `new economics of science', the production and distribution of knowledge are analysed from the point of view of information disclosure (cf. Dasgupta and David, 1994). The central question is whether knowledge is pursued in order to increase the public stock of knowledge, or to generate rents from its private exploitation. From this perspective, two kinds of systems may be distinguished, usually referred to as *Science* and *Technology*. In the former, knowledge is to be published openly, while in the latter, results are to remain a secret. This distinction between regimes can also be arrived at by focusing upon market mechanisms. Technology is the realm of the market, supported by intellectual property rights (IPRs) as granted by the state. Science, on the other hand, is a regime created by the state in an effort to correct market failure by granting subsidies and creating public laboratories. As Dasgupta and David stress, knowledge workers may be `scientists' or `technologists'—or both. It is not their cognitive practices, but the regime in which they work that will decide upon the matter.

In this article, a specific kind of knowledge will be analysed: software. Under what kind of market and non-market modes are computer programs being developed? In line with the analysis above the distinction between public disclosure and private appropriation of knowledge will be the central focus. IPRs will feature prominently in the analysis, surprisingly in *both* regimes. In the case of software, the market regime is in force within companies, mostly producers of hardware or software. The non-market regime, somewhat unusually, obtains within communities of computer hackers, who can be found anywhere, both inside *and* outside universities.

Therefore, in order to avoid misleading connotations, these two software regimes will no longer be referred to as Technology and Science, but as *Private* and *Public Regimes* respectively.

Software merits special attention, as it has unique qualities that set it apart from other types of knowledge products. A programmer starts with an idea that is specified in an algorithm. It is this algorithm that can be programmed. In the early days of computing, this took place directly in a language that the computer could read (machine language). Such languages, however, are difficult to `read' by human beings, let alone to change. Therefore, higher order languages, more easily readable, were invented to make programming easier. Since then, programming is done in a computer language, which subsequently has to be transformed into a machine language. In computer jargon: *source code* has to be translated by a compiler into *object code*. It will become clear that this distinction between algorithm, source code and object code plays a vital role in both regimes.

First the Private Regime as it obtains within companies will be analysed. It will be shown that from the 1980s onwards, both secrecy and IPRs have evolved considerably. Next, the analysis focuses on the Public Regime of hackers freely sharing source code, which has evolved alongside. IPRs will be shown to play a rather different role here: as copyright holders, authors created new kinds of licenses that regulate the inclusion of others (instead of excluding them). From the 1990s onwards, this movement for `open source software' has also made inroads into the private sector: by way of experiment, some firms took a free ride on existing projects, or opened up software projects of their own to outside hackers. The analysis will show that, as a result, open as well as mixed property regimes evolved (combining elements of both the Private and the Public Model), and new open source licenses were formulated in order to accommodate business interests.

Within the academic community, open source software development is increasingly attracting attention. For purposes of comparison, the following sources should be mentioned in particular. First and foremost several studies by Yochai Benkler and Lawrence Lessig, legal scholars who opened up valuable avenues for research on open source software (Benkler, 2001, 2002a, 2002b; and Lessig, 2002a, 2002b). More recently, Research Policy published a special issue on this matter (Vol. 32, No. 7, 2003), in which several authors touched upon the subject of property rights. From these, West (2003) in particular is relevant for purposes of comparison. All of these sources I will refer to later at several instances. On beforehand, it would seem useful to mention on what account my analysis is different. Broadly speaking, concerning the matter of IPRs, I do not restrict myself to the simplified picture of the open source community as using only two types of license (either the General Public License or the Berkeley Software Distribution license; cf. below), but I explore more fully the whole range of `open source licenses' that evolved, whether drafted by individuals, organizations or companies. Moreover, the complexities arising from combining source code with different licenses are explored. As a result, a more fine-grained picture will emerge of the open source movement and its communities of users.

2. Private regime

2.1 Protection of intellectual property

In order to develop new products and/or processes, companies have to invest in research and development (R&D). These investments, however, are tricky: the fruits of them may easily be expropriated and/or imitated by competitors. Companies, therefore, have no choice but to protect

their competitive advantage from R&D. The exploitation of newly developed intellectual assets has to be safeguarded, whether by legal or institutional arrangements. Thus, the Private Regime hinges critically upon *excluding* others from a firm's intellectual assets (Liebeskind, 1996).

A variety of protective mechanisms are in use. In a series of studies that have been conducted over the last decades it has been established, that these cluster into three main strategies of intellectual property protection (Levin et al., 1987, the `Yale survey'; Harabi, 1995; Cohen et al., 2000, the `Carnegie Mellon survey'; Cohen et al., 2002): (1) relying on lead time and manufacturing/service advantages over competitors; (2) stressing secrecy of all aspects of R&D; (3) patenting of obtained inventions. Their use is not completely a matter of free choice, but depends on the specific context and character of R&D. From the Yale and Carnegie Mellon surveys it can be deduced that in the last two decades the importance of both patenting and secrecy has grown remarkably (Cohen et al., 2000, pp. 12-13). In terms of effectiveness, lead time and secrecy rank higher than patents for both products and processes (both in the US and Europe) (Cohen et al., 2000, pp. 9-10). It has to be remarked, moreover, that these strategies are usually combined. For example, the patenting strategy appears never to be used in isolation (Levin et al., 1987).

Below, I will analyse the ways in which these three mechanisms are actually used by software firms to protect their source/object code, especially as they change over time. Note that someone like Benkler (2001, 2002a) uses them as tools in distinguishing strategies for the production of information generally. In effect, he *combines* the three protection mechanisms: while his `Mickey' and `romantic maximizer' strategy rely on patenting (or, more generally, on IPRs), his `quasi-rent seeker' strategy relies on *both* lead time and secrecy. I prefer to keep the three protection strategies separate, while, as remarked above, many combinations will appear in

practice.

About the lead time strategy not much needs to be said. In software development, it would seem to have been always in use. Firms rush to implement new features or invent new systems, in order to stay ahead of the competition. Many important `battles' between software producers bear witness to this strategy: whether it is Word vs. Word Perfect, Microsoft Internet Explorer vs. Netscape Communicator, or any other. Over the years, however, its essential features have remained the same. As to the other two strategies, of secrecy and patenting, a closer analysis is needed, while in the last two decades these have been extended and adapted to the specific context of software.

2.2 Secrecy

In the early days, software was tied to hardware. The company (like IBM or Digital) that sold this combined package, as a rule took care of maintenance also. Software as a special product did not exist as yet. Accordingly, protection was never an issue. This started to change in the 1970s, as hardware and software were `unbundled' and turned into products of their own. A decade later, the personal computer was created and started to sell in the millions. In the slipstream, software was about to become big business. How to fight imitators and guarantee a continuing stream of income from software?

As a first protective step firms decided to sell software in object code only. As a result, customers can hardly know what the software is all about, let alone change it according to their own wishes. This is so, while machine language can hardly be interpreted properly by a programmer. Moreover, the road back, from object code to source code (decompilation), which would allow

proper interpretation, is hardly feasible. In that sense, software in object code may be said to have been turned into a black box: not to interpreted, not to be changed.

Next to this *technical* means, companies considered *legal* means to keep their software a secret. Actually, they would have preferred to pursue IPRs as a way of protecting their intellectual assets. But the courts were not (yet) very hospitable to granting either copyright or patent protection to any aspect of software (cf. below). Therefore, *trade secrecy* laws became companies' first legal choice. According to these, information may qualify as a trade secret provided that (a) the information derives economic value from remaining a secret, and (b) the firm takes pains to keep it a secret. In two ways, these laws were used as a means to protect software (cf. Johnson, 2001, ch. 6).

Trade secrecy has first and foremost been employed to tie *software developers* down. Incidentally, this is the staple way for all employers, whatever their branch of business, of protecting their intellectual assets. As a matter of routine, employees have to sign clauses of confidentiality, in which they agree not to disclose company information. Both in their present job, and if they quit, for a period of some years afterwards, employees are obliged to keep secret confidential information of economic value in general, and software in particular. This would presumably cover both object code and source code. Apart from the fact that nothing can prevent someone from starting to work elsewhere on the general *ideas* of the software involved, this solution has met—and still meets—a curious fate. Confidentiality clauses do have a place in (US) law, and (US) courts do enforce them. Nevertheless, if the experiences of Silicon Valley concerning employees in information technology apply more generally, companies turn out to be hesitant to try to *enforce* these clauses (for the sequel, cf. Hyde, 1998). What are the reasons firms as matter of routine ask their employees to sign such clauses, but as a rule never bother to

enforce them if need be?

First and foremost, such court cases mostly fail. According to Hyde's findings (1998), juries and judges simply dislike suits against departing employees. Only in rare cases, such as when information stored in material form is stolen (documents, diskettes), someone is convicted. The background for such reticence may be economical: most observers agree that the economic success of Silicon Valley hinges mainly upon the `high velocity' labour market in which employees switch jobs frequently, thereby spreading around knowledge and information.

But there is another reason why companies have remained hesitant to appeal to the courts (Hyde, 1998). Whatever the outcome, they are sure to acquire a bad reputation of being a bully towards their (leaving) employees. Such a reputation can be very harmful in attracting young talent that wants to be unfettered by contractual ties. Compare the famous lawsuits of Intel (1989) and IBM (1991) against ex-employees, accused of taking trade secrets to a competitor. Both suits were lost, and subsequently backfired by creating a nasty public reputation for the firms involved.

Next, software firms targeted their *users*. Upon obtaining a copy, they were to signs secrecy clauses. This is, of course, an exceptional measure. No user of any other kind of invention has to swear secrecy. Whether an automobile, a drug or a DVD player, there is no need for such clauses. The decisive difference is, of course, the instant reproducibility of software: it can be copied in an instant, at almost no cost. With the advent of the Internet, it can also be spread all over the globe. In order to fight this nightmare of software `piracy' (as it is called), software lawyers turned to trade secrecy laws. For tailor-made software, the client was made to sign confidentiality clauses. For mass-distributed software, firms invented the `shrink-wrap' license: upon unwrapping the software, users automatically comply to licensing terms that forbid to copy or

distribute the software any further (cf. Branscomb, 1994, ch. 8). The validity of the latter legal `invention' has always been doubtful: is this really a contract between equal partners?

More importantly, a subtle nuance throws doubt upon the whole approach of relying on trade secrecy (cf. Johnson, 2001, ch. 6). If software is tailor-made for a customer, (s)he will usually require to obtain the source code, in order to be able to modify the software if necessary; this undermines the second basic condition for information to qualify as a trade secret, viz. keeping it a secret. Similarly, although only object code is provided, selling software on a mass scale does much to annihilate trade secrecy in a formal sense. In practice, therefore, this protection has never acquired much force. Another circumstance for it to fade into the background probably is, that later on stronger and more reliable instruments of protection came to the fore: copyright and patenting.

2.3 Copyright and patenting

So much for the strategy of secrecy. While still in use, from the 1980s onwards firms started to push for vesting property rights in software (for the general line of argument below about copyright and patenting of software, cf. Johnson, 2001, ch. 6; for more details, cf. Samuelson, 1990). Referring back to the Levin/Cohen typology of intellectual property strategies: a patenting strategy—or, more generally, an IPRs' strategy—unfolded. This is mainly the case for the US, in other continents the developments are `lagging behind'. Below, therefore, I will concentrate on developments in the US.

Company lawyers soon met with success in applying for *copyright* protection of software. Copyright has actually been developed for literary works like novels, plays and poems, and for

works of art like paintings and sculpture. It gives its creator all the rights of publication and distribution of the product in its *literal* form, extending far beyond his/her lifetime. The actual creative expression is thereby protected, not the underlying ideas. Note, that the critical issue for authors is not whether to ask for copyright or not; that will be obtained easily enough after fixing one's text on paper or disc. Rather, the issue is whether to sue a supposed infringer or not. That will cost money and effort.

Gradually legal institutions in the US became favourable to copyright claims for software. This has meant in practice, that both object code and source code came to enjoy protection; copyright has turned into an effective deterrent against software piracy. But neither the underlying ideas nor the algorithm as a whole were protected by copyright. Firms soon realized, therefore, that what is most valuable in software, going beyond the literal text, was still unprotected. So they started to push for broader copyright protection, encroaching upon the domain of underlying ideas. In a range of lawsuits, the so-called `look and feel' of software was tested out. This comprises aspects such as commands, icons, screen layout, screen sequence, and user interface functionalities (Samuelson and Glushko, 1990).

At first, these efforts met with success. After 1990, however, gradually the tide turned, and most previous `gains' were lost. Courts took a more critical stance and expressly limited copyright to the verbatim text of (object/source) code. They adopted the position that software features only qualify for protection if they are merely of aesthetic value. If they are considered to be functional features (as well), copyrights are not granted. The famous 1990s' lawsuit that Apple finally lost against Microsoft (about its graphical user interface) marks this turn around. Since then, the reach of copyright protection did not change much anymore. In spite of these changes in legal status, copyright has remained—and still is—the standard way of protecting one's software

packages.

Subsequently, American firms explored the other imaginable option for software: *patenting*. In general, patents are granted for inventions. They are sought routinely for the fruits of industrial R&D. In particular, to obtain a patent an invention must be useful, novel and non-obvious; moreover, it should be found statutory subject matter (as a machine, article of manufacture, or process). If a patent is granted, the inventor obtains the right to exclude others from making, using or selling the product; if others want to exploit the invention, they have no choice but to try and obtain a license. A patent lasts for 20 years. In the case of software, in particular, this meant that developers started to present their inventions as implemented in the software for patentability, *not* the software itself. If protection is granted, it covers not only object code and source code literally, but also the underlying idea or algorithm.

At first, legal bodies were reluctant to accept software (or rather: software-related inventions) as statutory matter. Would that not imply that mental processes became protected, limiting the freedom of thought? Would it not also imply protection of mathematical algorithms, abstract ideas, scientific principles, or laws of nature, thereby obstructing the free development of science and technology? Both were explicitly forbidden by the courts. Gradually, these obstacles have been overcome. Starting with *Diamond vs. Diehr* (1981), but really accelerating from the 1990s onwards, patents have been granted to software in abundance.

Software patents now as a rule are claimed both as a process and as a machine; the latter kind of claims, `embodied' as they are, usually more easily pass the statutory test. It is instructive to see how the US Patent and Trademark Office now treats *process* claims. Let us peruse their 1996 guidelines, and see how they formulate matters (Patent and Trademark Office, 1996). Abstract

ideas are still the absolute threshold that cannot be passed. If a computer-related invention consists solely of mathematical operations, it will not be considered statutory. A demonstrable link with physical reality has to exist. This can be achieved in two ways. As a first option, the process may require physical steps to be performed outside the computer: either before the computer performs its operations (`pre-computer process activity'; e.g., measurement and analysis of electrocardiograph signals, *Arrythmia*, 1992) or after (`post-computer process activity'; e.g., controlling a robot with a collision avoidance memory, *In re Warmerdam*, 1994). As a second option, the process claim may be argued to be limited `to a practical application within the technological arts'. This is to mean that the software not only performs an algorithm in the abstract, but also uses it for a practical purpose; an example would be an algorithm modelling noise that is actually used to remove noise from a digital signal.

By now, US patent claims for software have grown into a flood: from almost 4000 in 1988, to almost 21,000 in 1999. Software patent grants have risen in a similar fashion: from 2000 in 1988, to about 20,000 in 1999. A rough estimate holds that more than 20,000 patents are currently granted yearly. On top are big companies like IBM, AT&T and Motorola. Notice, though, that elsewhere, e.g. in Europe, patenting is still the subject of heated debate.

Something remarkable has happened here. Copyright has been invented to protect works of culture, while patenting is meant to protect the fruits of science and technology. Usually, a work of creation is either the one or the other. In the case of software, companies tried to have it *both* ways (cf. Samuelson, 1993, p. 304). In the beginning (early 1980s), they stressed the literary aspects of software: it is a text written in computer language and/or machine language, worthy of copyright protection. Since that tactic, though successful, yielded only limited gains (verbatim protection of code), they changed their tune: software also performs, behaves, acts, transforms.

Therefore, it may also be considered a process, or a machine, and qualify as an invention (if useful, novel and non-obvious). This tactic also succeeded; this time, protection of a broader kind was the result. So currently the property rights of both copyright and patent may be invested in software. I can think of no other intellectual product to enjoy such double protection. No wonder, that some jurists, in order to circumvent this exceptional dual position, have argued for *sui generis* protection of software. It seems unlikely, though, that their opinion will prevail.

3. Public regime

3.1 Open source communities

At the same time, another quite opposite approach is germinating. The movement for `open source software' stresses the free flow of software in source code form.[1] The model advocated is one of a community of programmers, who freely exchange the pieces of software they produce. One asks each other for advice, comments, and fixes for bugs. Using the instruments of copyright or patent to protect one's intellectual assets as described above is considered improper and detrimental to software development. Note, that these programmers proudly call themselves `hackers'. To them, this word has positive connotations; whizz-kids causing havoc are denoted as `crackers'.

What are the advantages of such a model? Eric Raymond, one of open source's most ardent

[1]About the common denominator for this movement a lot of discussion has taken place. Next to the term `open source software' also the term `free software' is frequently used. The former is linked to economic justifications of source code sharing (cf. below, sections 3.1 and 3.4), while the latter is used by those stressing the freedom to cooperate that the model allows (cf. below, section 3.3). Throughout the article, I will use the more neutral term `open source software'.

protagonists, puts it as follows: `Given enough eyeballs, all bugs are shallow' (Raymond, 1997). If the open source process goes on and on, the code involved will grow ever better in quality. As a result, software reliability is greatly enhanced. No matter of small importance, as unreliability is considered one of the main defects of software in general. One basic condition should be fulfilled if this model is to work at all: developers should not restrict their sharing to object code (`binaries'). In such a closed form, amending and fixing software is just too cumbersome; it is almost a black box. Therefore, the original source code should be shared; hence the name of this movement.

Next to enhanced quality, open source gives control to the user: he/she is free to repair, modify, update the software as needed. While proprietary software, in binary form, keeps the producer in full control, open source software hands over control to users. In particular, this enlarged control gives the software `eternal life'. The use of commercial software will soon stop whenever the product is taken off the market. This is so, while the `key' has not been delivered to the customer. If, however, source code is made available, use may continue well beyond the commercial lifetime of the product. Experienced users know how to handle the software, and may continue to consult each other.

What motivates co-developers to contribute? A recent survey among developers on the sourceforge platform (which hosts open source projects for free) yielded the following answers (Lakhani et al., 2002). First and foremost seem to be the intellectual stimulus and the improvement of skills that writing code provides. Next are the felt need for the software (whether for work or for hobby), the desire to support the case of open source software, and the joy of working in a team. Lower on the list is the motive that one may obtain recognition from one's peers, upon which one's reputation and status are founded. These results are further confirmed by

a second, much larger hacker survey carried out by Rishab Ghosh and co-workers (Ghosh et al., 2002). Also here, learning and sharing of skills and knowledge, the desire to be part of the open source community, and improvement of software functionality were the main motivators reported. Less importance, again, is attributed to the improvement of job opportunities and the development of an open source reputation.[2]

Recognition and reputation may not be the prime motivators, hackers do care about them. No wonder, therefore, that in all open source projects credit is given to the original developer, and to those after him/her; the authorship of pieces of code and patches is made clear to every participant. The model, as can be seen, closely resembles the university publication system, in which disinterested scholars openly and freely comment upon each other's work, giving due credit (in the form of references) where credit is due.

After a slow start in the 1980s, the number of open source communities has grown steadily. At this time of writing, current projects in cyberspace are estimated to number over 60,000 (Freshmeat, 2005; Sourceforge, 2005). As a result, many reliable and high quality software packages have been produced. Their success has much to do with the development of the Internet, which facilitated open distribution and participation enormously. And in turn, some open source projects have yielded pieces of software that facilitate the running of the Internet (cf. O'Reilly, 1998). Important examples of the latter are: BIND (Berkeley Internet Name Daemon), which makes the domain name system (for www-addresses) work; Sendmail, which takes care of routing e-mail messages; and Perl, a scripting language predominantly used for web-server

[2]Because of a different methodology, a third larger hacker survey (David et al., 2003) can hardly be compared with the first two surveys. Nevertheless, it is surprising to find that in the survey items relating to supporting the case of open source software score highest of all as motivators to start `hacking', higher than reasons of improvement of skills or of software functionality.

software. All of these open source projects are still running.

The most successful stories of open source software are, of course, Linux and Apache. Linux goes back to 1991, when the Helsinki student Linus Torvalds wrote a new operating system for his 80386 PC. Over the years the number of participants, as well the size of the program, have grown at a tremendous rate. Nowadays, Linux is by far the biggest open source program available. The Apache project started in 1995, when a group of Californian web-masters came together to update the web-server software as originally developed by Rob McCool at the National Center for Supercomputing Applications, University of Illinois. Soon hundreds of hackers cooperated, and Apache became the number one server software on the Internet.

3.2 Licenses

What about the terms of disclosure of open source software? Is the source code simply put on a website, from which it can be downloaded by anybody? Actually, things are more complicated than that. IPRs do play a role here, though in an unorthodox fashion. Like with any piece of commercial software, open source authors claim copyrights in their products. As authors, they then proceed to write a license for future users of copies. Anyone downloading the software, is supposed to automatically comply to its terms (`click-wrap license').[3]

Within the open source movement, gradually a consensus has developed about what constitutes a proper `open source license'. After heated discussions the Open Source Initiative (OSI), created

[3]Note that, strictly speaking, running the software is always allowed. The license terms only speak to copying, modification and (re)distribution of the code. So a user is only bound by the license if (s)he actually does more than just run the downloaded package. In that sense, the term `click-wrap license' is slightly misleading.

in order to promote the cause of open source software in general, formulated a yardstick, called the `open source definition' (OSI, 1997-2005). A license conforms to this definition, if it allows to freely use, repair and modify the source code. It should also allow the program, corrections and modifications to be publicly redistributed for further use, repair and modification. So a continuous cycle of modification and public (re)distribution in source code is to be allowed explicitly. Moreover, everyone is to be included in the process; specific persons or groups may not be excluded by the license. Similarly, using the program is to be allowed for all purposes; the license may not discriminate against specific fields of endeavour (for all of these clauses, see OSI, 1997-2005).

Early on in the 1980s, two influential licenses have been formulated: the General Public License (GPL) and the Berkeley Software Distribution (BSD) license. These reflect fundamental differences of opinion among hackers about intellectual property and corporate interests. Since then, many more licenses have been written by open source hackers. All of them, however, take these two licenses as point of departure. It is worthwhile, therefore, to dwell somewhat longer on them, and on the historical circumstances in which they were created.

The hacker community is mainly of American origin, and developed along two branches since the early 1970s (the sequel is based on McKusick, 1999, and Raymond, 1999). The first hacker branch had its base at MIT's Artificial Intelligence Laboratory. Using PDP computers from DEC, they wrote their own operating system (ITS, for the PDP-10) and many application programs. From 1969 onwards they used the defense-built ARPAnet to freely exchange their software. The other branch consisted of hackers from Berkeley (University of California) and New Jersey (Bell Laboratories, part of AT&T). They created Unix, an operating system written in C-language that promised portability from one machine to the next. In the beginning, from 1974 onwards, it was

open source *avant la lettre*. Newly produced Unix versions (invariably called BSD) were distributed in source code to anyone who applied for a copy, for free, and freely redistributable.

In the early 1980s two important changes took place. For one thing, the East Coast branch had to switch to the PDP-11, and decided to switch to UNIX as well. As a result, both hacker branches started to converge. For another, in 1984 AT&T was split up. The company had always kept track which parts of the Unix code were actually written by its own employees. It henceforth started to enforce its copyrights for commercial purposes. Any Unix-user had to apply for a new kind of license: source code was still delivered, but it was no longer freely redistributable, and no longer without charge. So BSD-versions entered into a semi-commercial maelstrom. Hackers were dismayed, especially as license fees grew year after year. They decided to regain the tools that—they felt—were stolen from them. Two kinds of countermovement ensued.

3.3 General Public License

The most radical approach was developed by Richard Stallman, a programmer at MIT. In 1984 he resigned from his job and started the ambitious project of writing a free operating system from scratch (called GNU). It should be compatible with Unix, and ultimately replace this non-free operating system. The Free Software Foundation (FSF) was set up to further the project. At great speed Stallman proceeded to write several GNU pieces such as the GNU Emacs editor, the GNU Debugger and the GNU C Compiler, all of which were freely distributed by the FSF.

These releases came with a license called the GPL, also colloquially referred to as `copyleft' (FSF/GNU, 1989/1991) (Table 1). As any other open source license that was formulated later, the GPL allows source code to be freely used and modified, and to be freely (re)distributed. A

discussion permanente among hackers is allowed, even encouraged. However, the license comprises an intriguing invention: modifications and recombinations may only be redistributed under the *same* license terms as the original work. For any redistribution of (modified) GPL-ed code the GPL is mandatory. In the wording of the GPL (preamble): `If you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have'. Rights and duties are passed on from one recipient to the next in an endless chain.

This implies two things. For one thing, it is not allowed to deviate from the public path. Modifying the software and bringing it on the market *only* in object code with say the usual shrink-wrap license is forbidden. Such a commercial move is allowed, but only if a full source code version is publicly available as well. In this way, all modifications remain in public view forever. For another, it implies that GPL-ed code, from the original lines to its various modifications, always remains tied to the GPL. Once code has been GPL-ed, it remains so forever. Therefore the GPL has been compared to a `virus':[4] whenever a programmer turns to combining GPL-ed code with other code, the product as a whole (being `a work based on the Program') may only be redistributed under GPL-conditions.

Later on, the FSF also drafted a more `lenient' license. It was specifically written for use with libraries, which are collections of functions needed to write software (e.g., a C+-library). Suppose a library becomes GPL-ed. Then a new program that links with that library has to be licensed on GPL-terms as well. In order to create room for other (especially proprietary) kinds of licensing, the Library (or Lesser) General Public License (LGPL) was created (FSF/GNU, 1991/1999)

[4]The term `virus' was probably first coined inside Microsoft, in order to discredit open source initiatives (cf. Mundie, 2001). Despite its negative connotations, the term is commonly used nowadays in the literature about open source. Therefore I feel free to use the term as well.

(Table 1). It is quite similar to the GPL, but for the fact that it allows linking with a library, without dictating any license terms upon the new program. But, of course, modifications of the library *itself* still have to comply to the strict LGPL-terms.

Why did Richard Stallman formulate these (L)GPL regulations? Why this focus on keeping the code `free'? From reading his famous GNU Manifesto (Stallman, 1985/1993) it would seem that two motivations stand out. For one thing, Stallman seems to loathe IPRs, for software in particular. He argues that property rights are quite unnecessary for motivating programmers to write code; these are passionate enough as it is. Moreover, next to superfluous IPRs are also harmful: they inhibit the use and distribution of software. For another, Stallman considers the hacker scene to be a community within which sharing one's programs is `a fundamental act of friendship'. Copying software is `as natural as breathing'. The ideal to be pursued is a hacker community of friends in which source code is property held in common.

3.4 Berkeley

Next to the FSF, another kind of movement to set Unix free developed among programmers from Berkeley. Instead of writing a whole new operating system from scratch (the Stallman approach), they proceeded to `liberate' Unix files, libraries and utilities one after the other. Their approach was to check all files upon original `authorship'. If they were sure to be the sole authors, they felt free to (re)issue files on their own terms. If, however, files were a joint product of Berkeley and New Jersey programmers, they had no choice but to rewrite them from scratch. Based on this procedure, and with the help of many outside hackers, `freed' Unix files were issued one after the other. After a series of releases (starting in 1989), a complete Unix system (for 386 PCs) became available in 1992.

These releases came with a liberal kind of license: the BSD license, in use from 1989 onwards (BSD, 1998; cf. Table 1). Essentially, everyone may freely use, modify and distribute binaries and source code, in original or modified form. Everything is allowed, if only the original inventor is credited by retaining the copyright notice in all subsequent versions. Such a license, of course, allows everyone to take pieces of the code, modifying and adding wherever necessary, and sell the package as a closed commercial product only (i.e., without disclosing the source code). Exactly this, branching off the public path and exclusively taking the private path, had been disallowed by the GPL. In those early days of open source software, a number of other licenses were formulated upon the same model that were to be used widely since: MIT License, and Apache Software License. For practical purposes, as the differences are typically small, I will refer to these as a whole as `BSD-style' licenses.

Why this more liberal attitude? In their Unix time these Berkeley hackers had become convinced of the superiority of open source practices: it allows bug fixing, improving performance and adding new features freely (McKusick, 1999, p. 40). To them, open source simply yields better software. And good software was all they cared about. Therefore, they did not feel the need to add regulative clauses à la the GPL to their licenses: these would only inhibit the further spread of their software. Any redistribution, whether open or closed, was welcome to them. In this sense, they supported software both as a public *and* as a private good.

3.5 Spectrum

This describes the spectrum of licenses, as it emerged from open source's early days (the 1980s). Not surprisingly, many hackers and/or not-for-profit organizations were not satisfied with these

existing licenses, and sat down to write yet another license to accompany the release of their source code (companies, mostly entering the scene from 1998 onwards, did likewise; these are analysed below). Several organizations try to keep abreast of these developments. The OSI maintains an up-to-date list of licenses that they `approved' to conform to their `open source definition' (OSI, 2005). This certification program, by the way, had unintended consequences: as Bruce Perens remarked, it `inadvertently encouraged the proliferation of licenses' (Moen, 2000). Also the FSF, tied specifically to the cause of copylefted software, regularly reports about licenses currently in use, and comments upon them (FSF/GNU, 1999-2002).

What was the yield? I studied a dozen of such new licenses, whether or not they were conform to the `open source definition'; *all* licenses I could detect that accompanied published source code were analysed. I found that a number of new nuances were introduced: modifications may be freely distributed, but a request for a copy from the original maintainer is to be complied with; modifications may only be redistributed to the original developer; modified versions must be renamed; advertisements for the product should always mention all copyright holders involved; commercial distributions are not allowed; classes of users and/or purposes are excluded. Nevertheless, I would argue that a substantially new kind of license did *not* emerge. All of them are essentially variations on the same theme: BSD-like (most of them) or GPL-like (few of them). With the odd exception of course: the Hacktivismo Enhanced-Source Software License Agreement (Hacktivismo, 2002). It is a political manifesto even more radical than the GPL. While fully endorsing the GPL philosophy, it imposes political restrictions on use and modification of source code: the software may not be used to violate human rights, and neither destructive viruses nor code for surveillance purposes may be incorporated. Note that as a result, it does no longer qualify as a `proper' open source license.

At the same time, it should be pointed out that all of these `new' licenses just did not attract a following. They are hardly used by anyone else than their original designer, issuing his/her own piece of software on these terms. They never became sociologically significant: open source licenses (company-drafted licenses, to be discussed shortly, excluded) essentially remained dominated by the well-known (L)GPL and BSD-like licenses. This statement can be substantiated by statistics about the actual application of `approved' open source licenses. Although open source projects tend to appear and disappear quickly, and counting them in cyberspace is quite a task, available statistics, for whatever they are worth, all point in the same direction: the GPL is by far the most popular license. The Freshmeat open source software platform recently tracked about 36,000 packages; most popular were the GPL (70%), LGPL (6%) and BSD license (6%) (Freshmeat, 2005). If I combine the first two licenses into (L)GPL, and count licenses similar to the BSD (but excluding the `'new' ones just discussed) together in the category of BSD-style licenses, the results are even more telling: 77% belongs to the (L)GPL family, and at least 15% to the BSD-style family. In the public domain was just 1.3% of projects. Another open source software platform, sourceforge, provides similar figures (Sourceforge, 2005): more than 60,000 projects, with the GPL accounting for 68%, the LGPL for 11%, the BSD license for 7%. In terms of `families': (L)GPL licenses 79%, BSD-style licenses at least 14%. Just 2.6% of the packages were wholly in the public domain.[5]

The fact that most open source developers choose the GPL for their work can be explained as follows. The regulations of the GPL provide a guarantee that all subsequent contributions to and modifications of one's code remain a public affair; no one may take the code and develop a closed package from it that remains private. In that sense, `appropriation' of one's intellectual

[5]These freshmeat and sourceforge statistics apply to all projects undertaken in cyberspace, *including* those that carry open source licenses as written by companies (which will be discussed below).

assets is being prevented.

3.6 Public domain

The core of the open source software development process is public disclosure of its vital text: source code. In addition, it comes with a license that is intended to further an ongoing public discussion of the code that supposedly contributes to ever higher quality and more features. IPRs will be shown to play a supportive role here. This time not of exclusion of others (as in the Private Regime), but of inclusion of all that care to join.

In legal terms literary and artistic creations are either copyrighted, or in the public domain (meaning that free use of them is allowed). A work enters the public domain, either because the term of copyright expires, or because the author decides to renounce his/her rights. Open source software does not neatly fit into these categories. On the one hand, it is always copyrighted, hackers explicitly assert to be their authors; on the other, several kinds of public use are explicitly allowed. So open source software seems to be *both* at the same time.

In order to account for this paradox, I suggest that we no longer look at a work as a whole, but distinguish between its various uses instead. Each use separately, then, may be either reserved to owners (`rights of exclusion'), or allowed to anyone (`users' privileges'). The distinction between private right and public privilege applies to each and every use separately, not to the work as a whole. This approach, going back to Wesley Hohfeld, has been further developed of late by Michael Heller (cf. the account in Elkin-Koren, 2001, p. 196; further references therein). From this point of view, some products belong as a whole to the public domain (e.g., scientific data, or, of course, works with copyright term expired). But as a rule products belong to *both* categories:

copyright holders obtain rights upon some specific uses, but other uses are excepted by law (such as fair use) or fall outside the scope of copyright (such as reading a text).

This finer distinction is useful to better appreciate the case of software. Copyrighted software is such a `mixed good': copyrights obtain, but at the same time copyright exceptions apply (e.g, to decompilation for certain purposes), and uses outside the scope of copyright law are free (e.g., running the software). Seen from this angle, open source licenses can be interpreted as efforts at tilting the balance from private to public uses. The way in which this is achieved is remarkable: an exception to copyright law is created, not by law, but by *contract*. Open source licenses, at least the ones conforming with the `open source definition', have in common that the rights of publication and distribution of code as conferred by copyright become public privileges. At the same time, they stipulate regulations that users have to satisfy. The BSD-style license mentions very few regulations, while the GPL prescribes more of them (requiring, in particular, that modified code always remains open and GPL-ed). So open source licenses may be interpreted as creating, as it were, a kind of `regulated' public domain by contractual exception (which, obviously, differs from the public domain proper). With that, the usual role imputed to IPRs has been reversed. While in the Private Regime copyrights (and patents) are used to exclude others, in the Public Regime these are used for *including* others. This inclusion, however, is not a blanket one: copyrights specifically allow imposing *regulations* upon participating hackers.

Why do open source software authors not take to the usual solutions for handling property rights? One customary solution is to retain copyright's exclusive rights without exceptions. This is, of course, the usual practice for scientific publications. In academia, this scheme is satisfactory (apart from the question *who* obtains these rights): it allows (by lawlike exception) fair use, which gives scientific discussions enough leeway. At the same time, no more freedoms are

allowed (such as rewriting the text) because nobody really needs this option: science is better off with academics writing a whole new text than rewriting texts of colleagues. For open source software, however, this scheme would provide *too little* freedom. Code gets better, adepts believe, by rewrite en plein public; it needs that kind of collective effort. Therefore, open source licenses provide precisely that kind of freedom.

Another solution that suggests itself is to refrain from rights altogether and put the source code squarely in the public domain. Although some hackers do prefer this solution, the large majority does not: in their view, this would allow *too many* freedoms. The typical hacker does want to impose some regulations upon his fellow hackers. For one thing, as they care about their reputation, hackers want to remain recognized as the authors of their specific pieces of code. Therefore they write clauses requiring that copyright notices be retained in all subsequent versions of the software. For another, the more radical branch of the open source movement (as embodied in the FSF) is focused upon keeping source code free. In order to impose this condition upon users, `virus-like' regulations have been written into the GPL. All of these various clauses would be not enforceable without copyright asserted.

I first formulated this approach to interpreting the paradox of open source software in De Laat (2002, pp. 99-101). It may usefully be compared with other recent analyses. Benkler (2001, 2002a), on the one hand, introduced the `Joe Einstein' strategy (for information production in general), used by non-market actors to appropriate the benefits of their information outputs which are made freely available to the public. Open source communities fit the description. In Benkler (2002b), the author proceeded to develop a more specific and more elaborate model of hackers as practising `peer production', based on a commons of source code. This commons is open to everyone, but with regulated use: a `regulated commons'. These regulations are then cogently

analysed in terms of their functions for keeping the production of open source software alive. Lessig (2002a, ch. 4, and 2002b), on the other hand, provides an analysis of open source licenses, and concludes that the code commons is protected by a combination of contract law and copyright. In my analysis, the same notions turn up, but they become *connected* to each other: the open source movement creates and sustains a commons of source code open to any hacker, which is regulated by licenses that derive their legal force from copyright and contract law combined.[6] It is the complexities of this regulation, especially as these increase by firms accessing the commons, that will be explored more fully below.

4. Open source in industry

4.1 Incorporating the product

Gradually, as open source became more successful, corporations undertook to come to terms with the phenomenon. They perceived open source primarily as a threat: their corporate clients could be tempted to switch to open source products, being free after all. But they also felt challenged to incorporate some of the open source dynamics into their own business practices.

From the 1990s onwards, firms at first took open source *products* as given, and attached `free-riding' practices to them.[7] The following approaches can be delineated which, it should be noted,

[6]In order to avoid confusion, henceforth I will adopt the term `regulated commons' or `source code commons', instead of `a kind of regulated public domain', which was used above.

[7]Notice that although firms do take a free ride, they do not cause the usual `free-rider problem' of diminishing the value of the collective outcome involved. Actually, rather the opposite happens: the firms involved *contribute* to the cause of open source software, directly by providing services and additional software (though not for free; cf. below), and indirectly by increasing the size of the open source community (which is the source of further improvements of the

unfolded simultaneously rather than consecutively (cf. Hecker, 1998, who employs a somewhat different classification): (1) tying services; (2) readying one's hardware for them; (3) developing commercial closed applications on top.

(1) First, companies start to sell services to facilitate the use of open source software. Actually, many firms were set up with just that goal in mind. This may take many forms. Copies of programs can be distributed (either in source code or object code), say on CD-ROM or DVD, in order to facilitate its installation. Also, bug fixing, customizing, enhancements, consulting, and training of personnel can be provided. Finally, books and documentation about open source software can be sold. All of such companies I will refer to as `support sellers'. Sometimes they start to develop additional software; with these extra's, they try to distinguish themselves from the competition.

Some examples are the following. Around Linux, from 1995 onwards a whole free-riding industry sprung up. Next to Red Hat/Fedora and SUSE (now part of Novell), Knoppix, Mandriva, and Xandros remain today as some of the main distributors of Linux services. In a similar fashion, Apache services are provided by Covalent Technologies and C2Net Software (now part of Red Hat). And O'Reilly has managed to become the distributor par excellence of state-of-the-art books about open source software.

(2) In addition, established hardware firms take measures to let open source software run on their existing products. For the purpose, enabling software (such as device driver code) has to be written. Examples in case of Linux include HP, SGI and IBM, soon followed by Compaq and Dell. Henceforth, clients may choose Linux on their computer systems (instead of say Windows

---

code available to all). No social dilemma is involved.

or Unix).

(3) As a third practice, companies take advantage of the broad base of free software, and proceed to develop and sell closed application packages that run on top of it. Until now, these consist mainly of already existing packages that are `ported' to Linux. After supporting this operating system on their computers, companies announced they would convert their existing applications to run on it too. Corel ported Word Perfect to Linux (1998), while IBM did so for Lotus (1999). Oracle and Informix joined by porting their database software to Linux (1999). All of this is commercial software: in object code and for a fee.

What kind of property regime applies, whenever *new* code is being produced? Often, free-riders simply stick to their old ways, and continue a closed regime. Supporting software (approach 1), enabling software (approach 2) or applications (approach 3) come with a commercial license. Sometimes, however, another approach is taken, and a *public* regime is tried out: supporting or enabling software are opened up with an open source license. Let me illustrate this statement by showing the different ways in which support sellers of Apache web server software may release applications on top of the free base (which comes with a BSD-license). On the one hand, a firm may release the extra software as binaries with a commercial license (notice that, in the process, selling support—approach 1—and selling applications—approach 3—merge together). A case in point is Covalent Technologies with its `Enterprise Ready Server' (binaries, $ 1495); this is Apache with some proprietary software for web infrastructure management on top. On the other hand, a firm may add software in source code, with an open source license attached to it. Take C2Net Software, that sells support for the `Stronghold Secure Web Server' (from $395 onwards). This is Apache, with extra modules that are based on the OpenSSL, SSLeay and mod_ssl projects. After these had been proprietary at first, the company preferred to open up these

projects: they became open sourced on the same license terms as Apache.

Usually, support sellers have this choice, of providing add-ons either on open or on closed terms. Notice, though, that this is not so if the software base comes with the regulatory GPL, and the enhancements are closely connected with it. Then the `virus' character of the GPL leaves hardly any choice but to attach a GPL to the extra's as well. A case in point is Red Hat, which sells `Red Hat Linux' on CD-ROM or DVD with documentation and support. Apart from compiling and testing for performance and reliability, the firm also develops new programs. An example is the Red Hat Package Manager (RPM), that simplifies the task of installing and removing applications on top of Linux. Red Hat has attached the same license as applies to the Linux kernel (the GPL), while the RPM `interacts sufficiently' with the kernel `to make any other licensing scheme difficult' (Red Hat, 1995-1997, par. I.3). An additional reason for choosing the GPL is, that their managers endorse the points of view of the FSF; to them, software should be free (cf. Young, 1999).

The overall conclusion may be drawn, that support sellers sometimes contribute software themselves. Then, a public regime of software development may be tried out, either voluntarily, or semi-compulsorily (if the enhancements interact closely enough with GPL-ed files in the free base). Such open regimes are the first instances of corporations actually adopting the hacker model of software development. On a small scale, hacker logic has entered the firm.

4.2 Incorporating the process

After these modest beginnings, of free-riding upon open source products, firms took a closer look at the *process* of open source. They started to open up, by way of experiment, some of their *own*

software projects to the outside world; opening up no longer remained confined to additions to software created by hackers outside of the company. The source code was put on a public website, for any hacker to load down. Moreover, further development of the software was actually to take place in the open: the website involved became a public software laboratory. The first company that ever did so on a grand scale was, of course, Netscape: in March 1998, it put almost the entire source code base of its experimental webbrowser, Communicator 5.0—numbering about two million source lines of code—, on the Internet. Soon after, other firms followed suit. Apple, AT&T, Borland, HP, IBM, Intel, Nokia, and SUN, as well as dozens of smaller firms, initiated similar experiments (cf. West, 2003; De Laat, 2004).

How to make money with such a move? Actually charging a fee for the open source package would not constitute much of a `business model', for two reasons. Any buyer may start to resell the software for a lower price, or even put it on a website, available to everyone at no charge. In addition, asking money for the software would severely hamper the development of a dedicated community of developers. Instead, companies focused their efforts on mobilizing as many hackers as possible to download the code—at no charge—, and to return comments, bug fixes and modifications. A thriving open source community was to be created that would produce ever improving software. On the assumption that a broad base of users/contributors develops, basically three ways to make money evolved, which mirror the discussion in section 4.1 above: selling (1) support, (2) connected hardware, and (3) commercial software on top. So, I would argue, whether one takes a free ride upon open sourced software from *someone else*, or opens up *one's own* software, the ways to make money are basically the same.

(1) To begin with, companies, as connoisseurs par excellence of their public source code, are in the best position to offer various kinds of services to users: documentation, consulting,

customizing and training of personnel. So, in effect, companies turn into `support sellers' of the source code they have created in the first place. This is a model with a fully open regime for one's software, both base and enhancements (à la Red Hat selling Linux, cf. above). An example in point is Zope Corporation, that in November 1998 opened up its main product, a web applications server called Zope. Complete source code can be downloaded for free from their website. Since then, many more applications, for content management in particular, have been open sourced as well. Around all of these projects thriving communities of contributing hackers have developed. Income flows from customizing solutions for customers, consulting, training and the like.

(2) Next, firms may turn to a public regime of open sourcing, with the ultimate goal in mind that it will enhance the sale of connected hardware. A recent example involves HP. It open sourced its e-speak technology, a platform designed to facilitate the delivery of e-services (January 2000). Their reasoning was, that if enough applications would be forthcoming, sales of supporting hardware (like servers and storage) would increase.

(3) Finally, firms may try to earn money by developing additional software on top of the freed base, and sell the whole product as fully supported closed binaries. After all, all open source licenses allow this distribution of source code compiled into binaries (as long as one does not `secretly' introduce modifications). Here we witness the introduction of a *mixed* property regime for one's software: open at the base, closed on top. An apt example is Netscape. Its policy had always been to release the browser proper, called Netscape Navigator, in several branded versions (binaries): Netscape Communicator Standard Edition (with extra's like Netscape Messenger for e-mail), Netscape Communicator Professional Edition (with even more extra's like Netscape Calendar). The more extra's, the higher the price. Netscape intended to continue this

policy with the open sourced 5th generation browser Communicator, and sell a whole range of branded versions with increasing degrees of sophistication.

Steps of opening up source code are not to be taken lightly. A calculus is involved here, of carefully comparing closed and open alternatives. In considering to open up source code, any firm has to ask itself: are we really going to make more money in (1) services, (2) hardware and/or (3) commercial software that build upon a freed and therefore expanded user base, than by simply sticking to our software and selling it all on a commercial basis? As for the third strategy in particular, an additional tactical problem has to be addressed (cf. West, 2003): how much is to be opened up, how much is to remain closed? Where has the dividing line best be drawn?

Observe, that open sourcing one's software implies that the public domain—or, more accurately, the source code commons—is being embraced (as contractual exemption). However, while hackers do so for creating an ever ongoing cycle of public learning, companies have other motives. They do not access the commons for its own sake, but while the public learning cycle, presumably, will generate outcomes (like a massive user base and improved quality) upon which other activities may prosper: services, hardware and/or closed software. An *instrumental* view of the source code commons predominates. It may usefully be compared with the more general strategy of information production, called the `studious (or scholarly) lawyer', as introduced by Benkler (2001, 2002a). It is the strategy followed by individuals or organizations that distribute their information outputs for free, so as to be able to capture the correlated markets.

4.3 Licenses

In opening up source code, companies had to confront the issue of licensing. What regulations, if

any, did they attach to the uses of their code? Did they choose the licenses as pioneered by the open source movement, or prefer to write down new licenses? After all, unlike free hackers, companies have to keep a keen eye on their corporate interests. Some firms were content to pick and choose one of the existing, dominant licenses. Examples involve Borland (for its Kylix libraries) and HP (for its e-speak technology) that chose a (L)GPL. The majority of them, however, started to write their own licenses; their juridical departments could not resist the temptation. Judging from the lists maintained by the FSF and the OSI, some 20 new licenses were created, intended to accompany the source code that a firm divulged on the Internet. What were the results? Did it create anything new, and how significant these proposals became in sociological terms?

After reading all these texts, I found, like before, that much energy was invested in producing new types of regulations. In the main, however, existing malls were followed: the BSD-style license (very often; the Intel Open Source License, the Sleepycat Software License, and the Zope Public License are examples) and the GPL (very seldom; the Common Public License from IBM is an example). As exceptions to this rule two licenses, I would argue, stand out as new contributions. First the Mozilla Public License (MPL), together with its `twin sister' the Netscape Public License (NPL), both conforming to the `open source definition' (the subtle differences between the two will be analysed below). Netscape created these for its browser release in 1998. Both try to formulate a middle position between the GPL and the BSD license. Secondly, the Jahia Collaborative Source License (JCSL), created recently by the Swiss Jahia company, may be considered a new kind of license (Jahia, 2003). It is a reasoned exposé trying to find a centre point between open source licenses on the one hand, and proprietary licenses on the other. It allows free use for research purposes only, and institutes an exchange system of code contributions for royalties. As a result, although source code is provided, the license does *not*

conform to the `open source definition'.

However, as before, history is relentless. Almost all of these `company licenses' remain a footnote in history. Regulating the release of their master's software is their only function ever. They just do not catch on. As for BSD-like (or GPL-like) company licenses, firms `clone' the original: they are not inclined to use the originals themselves—let alone clones produced by other firms. Similarly, the novel JCSL is not in use by any other firm. In the sourceforge and freshmeat statistics, all of these licenses are below the .15 % mark, or even non-existent. As a consequence, the picture of dominance by the (L)GPL followed by BSD-style licenses has not changed.

However, one significant exception applies. Precisely the first ever `company license' that appeared, written by Netscape (the MPL in particular), must be considered a success. For one thing, many software packages in cyberspace carry MPL-terms, simply because the public browser experiment, started in 1998, is still running. For another, the MPL has become a model for other firm licenses: a new mould has been created. After substituting their own name for Netscape, companies simply copied its terms, either literally or semi-literally, and put their own name on top of the document (sometimes without even mentioning Netscape). Examples include: Apple, Borland (its Interbase Public License), Jabber, Lutris Technologies, Nokia, Ricoh and SUN (both the SUN Public License and the SUN Industry Standards Source License). As a result, we find both `original-MPL' packages and `clone-licensed' packages in cyberspace. Their numbers are just making themselves felt: the MPL proper covers 1.6% of sourceforge registered packages, while the MPL and its company clones included yield a percentage of 1.9%.[8] In absolute numbers: 958 packages are MPL-ed; with clones included the number becomes 1160

[8]For the freshmeat platform, the MPL-percentages are lower: .60 and .66 respectively (Freshmeat, 2005).

packages (Sourceforge, 2005). In comparison: the Apache Software License and the MIT License (members of the `BSD-family') apply to about 1100 projects each.

What are the contents of the N/MPL, and precisely why has it become fairly popular? Before going into the details of this, some background information is necessary about why Netscape decided to open source its browser at all.

4.4 Netscape

In 1994, Netscape had given away its Navigator browser, as binaries, for free, if used for non-commercial purposes. In response, Microsoft wrote the Internet Explorer, and distributed it, as binaries, for free to *anyone*. Thereupon, Netscape opened up the complete *source* code of its browser to the outside world (March 1998). This was the first major company project ever to become open sourced. The details are as follows.

In open source circles usually two kinds of releases exist alongside each other: a `developer release' (more up-to-date but in the experimental phase), and a `product release' (more mature and stable, considered fit for the average user). Netscape used a similar logic. Their product version, Communicator 4.0, which was already available for some time, was announced to become free of charge (to anyone). No source code however was released, as Netscape wanted to focus external help particularly on the developer version that they were working on. The source code of that version, Communicator 5.0, was put on a web-site created for the occasion. A special branch was created inside Netscape, the Mozilla group, in order to lead the whole open source effort, channel incoming contributions, decide on fixes and patches to the software, and release new versions regularly.

Netscape devoted much attention to the matter of licensing (the sequel is based on Hecker, 1998; Mozilla, 1998-2005; Hamerly et al., 1999). In view of their policy of releasing commercial editions in the future, what kind of license terms should be attached to downloading the free browser? At first the BSD-license was contemplated, which, after all, allows almost anything. However, it was considered too liberal: it does nothing to encourage users to donate back their modifications to the open source community. Therefore, Netscape's attention turned to the GPL, which requires any actual redistribution to be publicly available. They soon found out, however, that the `viral' character of the GPL would jeopardize Netscape's commercial interests: future modifications of the browser by outside hackers could impossibly be incorporated into their commercial software releases.

For one thing, consider the (future) branded versions of the Communicator 5.0 browser (binaries). These would preferably contain as yet those proprietary modules and cryptographic modules that had been removed from the browser just prior to its public release. As a result of incorporating future GPL-ed modifications into that commercial browser, Netscape feared that it would be forced to open up the whole product (on GPL-terms). Meeting those terms would be a heavy task, requiring rewriting as much proprietary code as possible, and removal of cryptographic modules (being prohibited by US law). For another, client and server software would have much source code in common between them; choosing a GPL for the client source code would force Netscape to open up their future server software (on GPL-terms). This was outright unacceptable to the firm.

Therefore company lawyers took to writing a new license: the *Netscape Public License* (NPL). Its core text is essentially similar to the GPL, and even more strict: all modifications one

composes *have* to be published as open source code, on the same NPL-terms. At the end of the license, amendments are formulated that grant Netscape special rights: it may at any time use source code from the public browser repository in its commercial products without having to expose the source code of these products (contrary to other NPL-terms). Moreover, Netscape may ignore NPL-terms, and attach license terms other than the NPL to that code (`alternative licensing'). One might paraphrase this proposal as a GPL for everybody—except for Netscape; to that firm BSD-like terms apply.

This license proposal was `beta-tested' in public, via a special web-site. Many hackers were enraged, especially by the special rights Netscape reserved for themselves. In response, the company decided to reduce the asymmetry involved. For smaller modifications of NPL-ed code conditions remained the same: these still have to be made publicly available (on NPL-terms). For a `larger work' however (combining original source code with new code, but where old and new code are organized in *separate files*), terms were relaxed: its creator was allowed room to distribute it as `a single product'. As for licensing, the old code obviously has to remain NPL-ed, while the new added code may be released under any other license (par. 3.7 of the NPL). Note that the connecting application programming interface has to be disclosed (NPL-ed) in order to enhance competition. So contributors no longer automatically work for the open source community and for Netscape, but may choose to exploit their greater findings all by themselves (as explained above, the LGPL had pioneered a similar kind of exception before). Effectively, the special rights for Netscape were limited now to smaller modifications.

The NPL, thus modified, was used for Netscape's browser release (version 1.0) (Mozilla, 1998b). However, the firm also created *another* license: the Mozilla Public License (MPL) (Mozilla, 1998a). The MPL is simply a reduced version of the NPL: the NPL *minus* the amendments

granting special rights to Netscape (Table 1). So for smaller modifications this MPL requires publication on the same terms (GPL-like), while hackers contributing major improvements may choose the type of license to go with them (for the *newly* written code, that is) (BSD-like). Netscape suggested that the Mozilla, striking the right balance between the more liberal BSD license and the more regulatory GPL, would be a fine license for such major browser contributions. In addition it suggested that the MPL was a license in its own right, applicable more generally as well beyond the specific context of Netscape's browser release. The company was right on both counts. Not only the MPL came to be widely used within the Mozilla project, it also became the mould upon which many other `company licenses' were drafted. As argued above, about a dozen other companies took up the Netscape proposals.

4.5 Incompatibility with the (L)GPL

This success was soon overshadowed by an unexpected drawback that came to light. The Netscape browser was not only a stand-alone product; its code could also be embedded in other software. However, developers wanting to incorporate N/MPL-ed code into a (L)GPL-ed package faced insurmountable licensing problems. An example is the Mozilla code for browser functionalities: it could not properly be incorporated into the GNOME project for developing a graphical user interface, a project that had already started under GPL-terms.

Why is this so? It is because both the (L)GPL and the N/MPL have regulative terms that clash with each other (cf. also FSF/GNU, 2001). Take a FSF-hacker that considers combining N/MPL-ed code with new code (a modification) or with new files (a larger work). Any modification of N/MPL-ed code has to be issued on the same license terms; so a GPL is not allowed. As for a larger work, while existing files have to retain N/MPL-terms, new files may be issued under any

other license. However, that other license cannot be a GPL, while GPL-terms would dictate that the *whole* product becomes GPL-ed. The same type of analysis applies when N/MPL-ed code is to be combined with existing code *already* under the GPL (a modification) or with existing files *already* under the GPL (a larger work); in both instances the terms of the N/MPL and the GPL clash as well. Finally, the same type of problems also emerge if instead of the GPL the Lesser GPL is taken into consideration (cf. in particular Mozilla, 2001-2003). The inescapable conclusion is that hackers preferring (L)GPL-style software development cannot in any useful way incorporate N/MPL-ed code.

In hacker circles this has been dubbed the *incompatibility problem*: the N/MPL is incompatible with the (L)GPL. Notice that among the main licenses—(L)GPL, BSD license, N/MPL—this is the *only* incompatibility. A BSD-style license is compatible with the (L)GPL because of its liberal terms (a combination of BSD-licensed code with (L)GPL-ed code may simply be issued under the (L)GPL); in the same vein, the BSD-license is compatible with the N/MPL. As about 80% of open source projects in cyberspace circulate on (L)GPL-terms, this incompatibility is a serious matter for the license as drafted by Netscape. Any code with a N/MPL will be shunned by the large majority of open source developers, since they cannot legally use it for development purposes (cf. Wheeler, 2002/2005). Using the metaphor of the regulated commons (Benkler, 2002b), this incompatibility means that *fences*, as it were, are erected between source code packages *inside* the commons. Users fragment into several communities, each with their own developmental privileges. While the BSD-community may use all code in the commons, the (L)GPL- and MPL-communities are barred from using code from each other.

Soon after the 1998 issue of the Mozilla browser on N/MPL-terms, Netscape decided that it would have to remedy the situation. As a first quick fix, in September 1999, upon moving from

version 1.0 to version 1.1 of the N/MPL, section 13 was added (Mozilla, 1999a, 1999b, 1999c). It allowed contributors of new files to the browser repository (whether Netscape or others) to attach one or more licenses of their choosing (`multiple licensing'). In order to prevent splitting of the code base (particularly into a GPL-ed base and a NPL-ed base), Netscape even expressly encouraged contributors to attach more licenses than just one (Mozilla, 1999c). In practice, this option was mainly used to attach a dual license: MPL + GPL.

This multiple licensing option remedied the incompatibility problem. However, only for the *new* files that came in. Already existing files still carried the single N/MPL. Thereafter, various options to eliminate the problem as a whole were considered by Netscape—and rejected (for an extensive discussion cf. Mozilla, 2001-2003). Releasing yet another version of the existing N/MPL, or composing an entirely new license were rejected, while unintentionally new legal problems might be created. Dual licensing with MPL + GPL was rejected, while it would not satisfy developers preferring to release on LGPL terms; similarly, dual licensing with MPL + LGPL would make life difficult for GPL-developers. Therefore, in the end a *triple licensing* scheme was chosen: the preferred format for *all* files was to become MPL + LGPL + GPL. Future developers would pick and choose the license(s) that suited them best: GPL-developers were to pick the GPL in order to create combined works, LGPL-developers were to choose the LGPL in order to incorporate browser code, and proprietary works were to be created by choosing the N/MPL (as before). This transformation would take time, of course. While the policy was declared in force with immediate effect for contributors of new files (2001), all creators of existing files (other than Netscape) had to be asked for permission to relicense their files in the repository according to the triple scheme.

Surprisingly, Netscape also announced that the Netscape Public License would be phased out in

due time. Why was the firm willing to give up the special rights it had so much struggled to obtain? The open source Mozilla project was to prepare the ground for the next generation browser product. Netscape indeed released a product version as late as November 2000, called Netscape 6 (the number 5 was cancelled); thereafter, Netscape 6.1, 6.2, and 7 appeared, all free of charge. However, the original plans to compose and sell more elaborate *commercial* editions (in binaries) never materialized. This is probably due to the fact that Netscape came in troubled waters: it was taken over by AOL, and lost the larger part of the browser market to Microsoft's Internet Explorer. So the special rights for Netscape, of creating modifications of source code without public release and of relicensing on other terms, were never exercised. While the original rationale had evaporated, Netscape decided to eliminate the NPL altogether.

But the Mozilla Public License remained, be it extended with other licensing options (the `triple license'). Obviously, MPL-clones are sure to encounter the same problem. No wonder that some firms that prefer such a clone for their releases have also taken to a multiple licensing scheme. In October 2000, SUN open sourced OpenOffice, an `office productivity suite' with applications for word processing, spreadsheet, presentation, graphics and database management. Components can be used as stand-alone products *and* as embeddable into other code. Therefore the MPL, SUN's preferred choice, would not do, and the corporation devised a dual licensing scheme of its own (SUN, 2000a). OpenOffice files carry a choice between the (L)GPL on the one hand, intended for use by (L)GPL developers, and the SUN Industry Standards Source License (SISSL) on the other, to be used by anyone else (SUN, 2000b). This SISSL was composed for the occasion, with terms quite similar to the MPL. Its relicensing terms are even laxer: *all* modifications of source code (also small ones) may be taken private, provided that certain specified standards for file formats and application programming interfaces are respected. If compliance is broken, a reference implementation of the modification must be sent back to SUN in source code; it will then be

added to the OpenOffice code base. In this way, open competition is maintained.

A similar change of license policy happened to the Qt graphical user interface toolkit, developed by Trolltech from Norway. The program became popular as a basis for the open source K Desktop Environment (KDE). However, the firm used to sell it with a commercial license (and still does), which aroused resistance within open source circles. As a first response, in 1998 Qt was *also* open sourced, with the Q Public License (QPL) attached to it, a self-made license, quite close to the asymmetrical NPL (Trolltech, 1999). However, the QPL was found to be incompatible with the GPL, still seriously hampering the use of Qt and KDE in GPL-circles (cf. Hecker, 1998). Therefore, after some more pressure by the FSF, two years later the company decided to attach a dual license to the open source release of Qt: QPL + GPL.

Potentially, of course, the problem of incompatibility applies also to the remaining, rarely used open source licenses. After all, it takes only one ill-considered clause in a new license and incompatibility (with the (L)GPL, or with the MPL for that matter) is the result.[9] In addition, mutual incompatibilities *among* the scarcely used licenses may emerge. In that case, even more fences are erected within the source code commons. A stark example is the Common Public License from IBM: as a copyleft license, it is *only* compatible with BSD-like licenses. Nevertheless, as about 95% of all open source licenses are (L)GPL, BSD-like or MPL-like, the above analysis exhausts the problem of incompatibility for the most part.

It may be concluded that no other viable open source license types exist other than the (L)GPL

[9]The FSF, committed as it is to the cause of free software, scrupulously keeps track of all open source licenses currently in use and, based upon their interpretation of compatibility, labels them as either compatible or incompatible (FSF/GNU, 1999-2002).

and the BSD license. The MPL can hardly stand on its feet alone as a single license, because the `viral' nature of the (L)GPL bars use of MPL-ed code or MPL-ed toolkits in (L)GPL-circles for purposes of development (and vice versa). Releasing source code under such terms introduces a fence in the commons between (L)GPL-ed and MPL-ed code, fragmenting users into communities with differential privileges (namely: BSD-, (L)GPL- and MPL-communities). If developers want to make full use of the creative energies of *all* potential visitors of the commons, only the (L)GPL and the BSD license (either on their own, or as part of a multiple license) are to be advised for their source code releases.[10]

This analysis clearly shows that the (L)GPL is not only the most widely used open source license, but also, precisely because of this, imposes conditions on anyone contemplating terms upon which to open up his/her source code. How does the proposed license compare with the (L)GPL? Can the source code be used by (L)GPL-developers without running into legal contradictions? The political program as formulated by Richard Stallman 20 years ago may in this respect be called a success. Not least by his continuous insistence on copyleft, most open source software packages have actually become copylefted, and as a result, the (L)GPL calls the tune to all other types of open source developers. This has an important function for keeping the `peer production' of open source software alive; to wit, both private appropriation and potential fragmentation into communities with differential user privileges have largely been held at bay. The source code commons may be imagined as having a fence almost all *around* it (the (L)GPL), but still almost no fences on the *inside*.

[10]No wonder that the OSI, complaining of the ever expanding number of open source licenses, works on proposals that will cut down the number of licenses effectively in use to a handful (including at least the (L)GPL and the BSD license) (LaMonica, 2005).

5. Summary and conclusions

In this article the two main kinds of property regimes for software development have been described and analysed. On the one hand, a Private Regime obtains within corporations, using the strategies of secrecy and of copyright and patent to protect their intellectual assets from imitation and expropriation by others. So the core of this regime is *exclusion* of outsiders. These strategies of protection have been strengthened in the last decades, in a very distinctive fashion. As for secrecy, apart from requiring this from their employees, software producers went ahead and asked the same from their users. As for the patenting strategy, from the 1990s onwards software patents have been granted in abundance, at least in the US. Moreover, already earlier than that, copyright had been developed into a legally acknowledged way of protecting software code verbatim. By adding the twin sister of patenting, the patenting strategy proper has evolved into a full-blown IPRs' strategy.

On the other hand, a Public Regime has evolved in hacker communities, in which source code is freely exchanged and discussed. While retaining copyright, authors of software formulate so-called open source licenses that transform their private copyrights into public privileges by contractual exception. The terms of the license effectively regulate public uses of the code. So property rights are not used for excluding others, but for *including* them, be it with some amount of regulation. A `regulated commons' is created. The first licenses ever written, the (L)GPL and the BSD-style licenses, have remained dominant from the beginning, and currently cover about 80% and about 15% respectively of all open source packages.

In order to come to terms with this phenomenon, corporations at first took a free ride upon

existing open source software by selling support, connected hardware, or closed applications on top. Interestingly, support sellers that developed additional software sometimes chose a Public Regime for these enhancements (e.g., Red Hat); almost silently, the open source model had entered the corporate world. Thereafter, from 1998 onwards, firms started to open up some projects of their own to the outside world. In these initiatives, a Public Regime was applied, quite similar to the one in hacker circles. This was not philanthropy; the business models behind them, closely mirroring the strategies discussed above, were selling of support, connected hardware or additional closed software on top. In these instances the source code commons is not accessed for its own sake, but as a means of generating revenue in connected ways. In the case of selling software on top in particular, we witness the innovation of a *mixed model* of property relations: a Public Regime for the base, and a Private Regime for the enhancements, all being carried out within—or at least supported by—one and the same firm.

So, effectively, the Public Regime has been incorporated into corporate logic. A new type of exploitation strategy of intellectual assets has been born. Besides exploiting them *directly*, by protecting them from others (Private Regime), firms may also exploit them *indirectly*, by sharing them first (Public Regime). The assertion referred to at the beginning of this article, that all strategies of knowledge appropriation depend critically upon the exclusion of others (Liebeskind, 1996), must be qualified; at least for software, the *inclusion* of outsiders may usefully become a part of them. The commons may become a basis for future profits.

As for licensing, firms mostly created their own license. All the while, guarding their business interests was a predominant concern. From all of these initiatives, only one viable new type of license emerged that also became a model for other company licenses: the Mozilla Public License (MPL). Developed by Netscape in order to accompany its browser source code release in

1998, it essentially occupies a middle position between the (L)GPL and the BSD-style licenses.

As a result of the open source movement, the spectrum of available software licenses has been extended. Software used to be sold on shrink-wrap license or contractual terms; or, after rights had been waived or had expired, it ended up in the public domain. Open source licenses have actually opened up a new terrain *between* both extremes (Table 1). The hacker community took the first steps by inventing the (L)GPL and the BSD license. Thereupon, the move into business prompted company-specific licenses to be invented, from which only the MPL emerged as modestly successful.

Soon it became apparent, however, that the MPL is incompatible with the (L)GPL. Choosing a MPL thus amounts to making one's source code irrelevant, for development purposes at least, to 80 % of the hacker population. A fence, as it were, would be erected inside the source code commons. In order to remedy this problem and effectively connect with developers of all persuasions, Netscape had to broaden the MPL with other licensing options and attach a multiple license (including both the LGPL and the GPL). In general it may be concluded that in releasing source code, developers are best advised to choose the (L)GPL or a (L)GPL-compatible license (like the BSD license). If, nevertheless, an incompatible license is preferred, a multiple licensing policy including the (L)GPL (or a compatible license) is recommended. In this way the gap with GPL-developers will be bridged, the active participation of all hackers ensured and fragmentation of users/developers avoided.

With this, of course, the analysis has only just begun. At the moment, open source in industry is still a marginal phenomenon, the vast majority of their software is released on closed terms. Will open source grow in importance as a new model of developing software? More generally, is `peer

production' of code the best way of developing reliable software, and a closed regime only strengthening monopolies that harm the public at large? Or, alternatively, is a closed regime a vital condition for the software industry to prosper, and open source a mistaken conception that harms business interests? All of these questions remain to be answered.

INSERT TABLE 1 ABOUT HERE

References

Benkler, Y., 2001. A political economy of the public domain: Markets in information goods versus the marketplace of ideas. In: Dreyfuss et al., pp. 267-292.

Benkler, Y., 2002a. Intellectual property and the organization of information production. International Review of Law and Economics 22, 81-107.

Benkler, Y., 2002b. Coase's Penguin, or, Linux and *The Nature of the Firm*. The Yale Law Journal 112, 369-446.

Branscomb, A.W., 1994. Who Owns Information? From Privacy to Public Access. BasicBooks, New York.

BSD, 1998. The BSD license. Available at http://www.opensource.org/licenses/bsd-license.php, accessed 24 February 2005.

Cohen, W.M., Goto, A., Nagata, A., Nelson, R.R., Walsh, J.P., 2002. R&D spillovers, patents and the incentives to innovate in Japan and the United States. Research Policy 31, 1349-1367.

Cohen, W.M., Nelson, R.R., Walsh, J.P., 2000. Protecting their intellectual assets:

Appropriability conditions and why U.S. manufacturing firms patent (or not). National Bureau of Economic Research, Cambridge, Massachusetts: Working paper 7552.

Dasgupta, P., David, P.A., 1994. Toward a new economics of science. Research Policy 23, 487-521.

David, P., Waterman, A., and Arora, S., 2003. FLOSS-US: The free/libre/open source software survey for 2003. SIEPR, Stanford University, California. Available at http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf, accessed 17 June 2005.

De Laat, P.B., 2002. Eigendomsrechten op software: copyright annex patent of open source? [Property rights in software: copyright plus patent or open source?] Krisis, Tijdschrift voor Empirische Filosofie 3(2), 89-107.

De Laat, P.B., 2004. Evolution of open source networks in industry. The Information Society 20, 291-299.

DiBona, C., Ockman, S., Stone, M. (Eds.), 1999. Open Sources: Voices from the Open Source Revolution. O'Reilly, Sebastopol.

Dreyfuss, R.C., Zimmerman, D.L., First, H. (Eds.), 2001. Expanding the Boundaries of Intellectual Property: Innovation Policy for the Knowledge Society. Oxford University Press, Oxford.

Elkin-Koren, N., 2001. A public-regarding approach to contracting over copyrights. In: Dreyfuss et al., pp. 191-221.

Freshmeat, 2005. Statistics and top 20. Available at http://freshmeat.net/stats/#license, accessed 28 January 2005.

FSF/GNU, 1989/1991. GNU General public license. Available at http://www.gnu.org/licenses/gpl.html, accessed 24 February 2005.

FSF/GNU, 1991/1999. GNU Lesser general public license. Available at

http://www.gnu.org/licenses/lgpl.html, accessed 24 February 2005.

FSF/GNU, 1999-2002. Various licenses and comments about them. Available at http://www.gnu.org/philosophy/license-list.html, accessed 24 February 2005.

FSF/GNU, 2001. Frequently asked questions about the GNU GPL. Available at http://www.gnu.org/licenses/gpl-faq.html, accessed 24 February 2005.

Ghosh, R.A., Glott, R., Krieger, B., and Robles, G., 2002. Free/libre and open source software: Survey and study. Final report (D18), Part 4: Survey of developers. International Institute of Infonomics, University of Maastricht, The Netherlands. Available at http://flossproject.org/report/FLOSS_Final4.pdf, accessed 17 June 2005.

Hacktivismo, 2002. The Hacktivismo enhanced-source software license agreement. Available at http://www.hacktivismo.com/about/hessla.php, accessed 24 February 2005.

Hamerly, J., Paquin, T., with Walton, S., 1999. Freeing the source: The story of Mozilla. In: DiBona et al., pp. 197-206.

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

Hecker, F., 1998. Setting up shop: The business of open-source software. Available at http://www.hecker.org/writings/setting-up-shop.html, accessed 24 February 2005.

Hyde, A., 1998. Silicon valley's high-velocity labor market. Journal of Applied Corporate Finance 11(2), 28-37. Larger version available at http://www.andromeda.rutgers.edu/~hyde/, accessed 24 February 2005.

Jahia, 2003. License FAQ. Available at http://www.jahia.org/jahia/page336.html, accessed 24 February 2005.

Johnson, D.G., 2001. Computer ethics, third ed. Prentice Hall, Upper Saddle River, New Jersey.

Lakhani, K.R., Wolf, B., Bates, J., DiBona, C., 2002. The Boston Consulting Group Hacker Survey. Release 0.73. Available at

http://www.bcg.com/opensource/BCGHackerSurveyOSCON24July02v073.pdf, accessed 11 February 2005.

LaMonica, M., 2005. Open-source board eyes fewer licenses. CNET News.com, 16 February. Available at http://news.com.com/Open-source+board+eyes+fewer+licenses/2100-7344_3-5578799.html, accessed 3 March 2005.

Lessig, L., 2002a. The future of ideas: The fate of the commons in a connected world. Vintage Books, New York.

Lessig, L., 2002b. Open source baselines: Compared to what? In: Hahn, R.W. (Ed.), Government Policy toward Open Source Software. AEI-Brookings Joint Center for Regulatory Studies, Washington, D.C., pp. 50-68.

Levin, R.C., Klevorick, A.K., Nelson, R.R., Winter, S.G., 1987. Appropriating the returns from industrial R&D. Brookings Papers on Economic Activity, 783-820.

Liebeskind, J.P., 1996. Knowledge, strategy, and the knowledge of the firm. Strategic Management Journal 17 (Winter Special Issue), 93-107.

McKusick, M.K., 1999. Twenty years of Berkeley Unix: From AT&T-owned to freely redistributable. In: DiBona et al., pp. 31-46.

Moen, R., 2000. A public discussion of open source licensing. First appeared in LinuxWorld.com. Available at http://www.linuxmafia.com/faq/Licensing_and_Law/licensing-discussion.html, accessed 24 February 2005.

Mozilla, 1998a. Mozilla public license version 1.0. Available at http://www.mozilla.org/MPL/MPL-1.0.html, accessed 24 February 2005.

Mozilla, 1998b. Netscape public license version 1.0. Available at http://www.mozilla.org/MPL/NPL-1.0.html, accessed 24 February 2005.

Mozilla, 1998-2005. Netscape public license FAQ. Available at

http://www.mozilla.org/MPL/FAQ.html, accessed 24 February 2005.

Mozilla, 1999a. Mozilla public license version 1.1. Available at http://www.mozilla.org/MPL/MPL-1.1.html, accessed 24 February 2005.

Mozilla, 1999b. Netscape public license version 1.1. Available at http://www.mozilla.org/MPL/NPL-1.1.html, accessed 24 February 2005.

Mozilla, 1999c. NPL version 1.0M FAQ. Available at http://www.mozilla.org/MPL/NPL-1.0M-FAQ.html, accessed 24 February 2005.

Mozilla, 2001-2003. Mozilla relicensing FAQ. Available at http://www.mozilla.org/MPL/relicensing-faq.html, accessed 24 February 2005.

Mundie, C., 2001. The commercial software model. Remarks made at the Stern School of Business, 3 May. Available at http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp, accessed 21 January 2005.

O'Reilly, T., 1998. Measuring the impact of free software. Available at http://www.ddj.com/documents/s=2861/nam1012433798/index.html, accessed 24 February 2005.

OSI, 1997-2005. The open source definition, version 1.9. Available at http://www.opensource.org/docs/definition.php, accessed 24 February 2005.

OSI, 2005. The approved licenses. Available at http://www.opensource.org/licenses, accessed 24 February 2005.

Patent and Trademark Office, 1996. Examination Guidelines for Computer-Related Inventions. No. 950531144-5304-02.

Raymond, E.S., 1997. The cathedral and the bazaar. Available at http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/, accessed 24 February 2005.

Raymond, E.S., 1999. A brief history of hackerdom. In: DiBona et al., pp. 19-29.

Red Hat, 1995-1997. Red Hat Linux 5.0: The official Red Hat Linux installation Guide. Available at http://www.redhat.com/docs/manuals/linux/RHL-5.0-Manual/user-guide/, accessed 21 January 2005.

Samuelson, P., 1990. *Benson* revisited: The case against patent protection for algorithms and other computer program-related inventions. Emory Law Journal 39, 1025-1154.

Samuelson, P., 1993. A case study on computer programs. In: Wallerstein, M.B., Mogee, M.E., Schoen, R.A. (Eds.), Global Dimensions of Intellectual Property Rights in Science and Technology. Office of International Affairs, National Research Council, pp. 284-318.

Samuelson, P., Glushko, R.J., 1990. Survey on the look and feel lawsuits. Communications of the ACM 33, 483-487.

Sourceforge, 2005. Software map. Available at http://sourceforge.net/softwaremap/trove_list.php?form_cat=14, accessed 28 January 2005.

Stallman, R., 1985/1993. The GNU manifesto. Available at http://www.gnu.org/gnu/manifesto.html, accessed 24 February 2005.

SUN, 2000a. The OpenOffice.org project: Foundations of office productivity in a networked age. Available at http://www.openoffice.org/white_papers/OOo_project/OOo_project.html, accessed 24 February 2005.

SUN, 2000b. SUN industry standards source license, version 1.1. Available at http://www.openoffice.org/licenses/sissl_license.html, accessed 24 February 2005.

Trolltech, 1999. The Q public license, version 1.0. Available at http://www.opensource.org/licenses/qtpl.php, accessed 24 February 2005.

West, J., 2003. How open is open enough? Melding proprietary and open source platform strategies. Research Policy 32, 1259-1285.

Wheeler, D.A., 2002/2005. Make your open source software GPL-compatible. Or else. Available at http://www.dwheeler.com/essays/gpl-compatible.html, accessed 24 February 2005.

Young, R., 1999. Giving it away: How Red Hat software stumbled across a new economic model and helped improve an industry. In: DiBona et al., pp. 113-125.

Abbreviations

BSD = Berkeley Software Distribution

FSF = Free Software Foundation

GPL = General Public License

IPRs = intellectual property rights

LGPL = Library (or Lesser) General Public License

MPL = Mozilla Public License

NPL = Netscape Public License

OSI = Open Source Initiative

Table 1
Rights granted to distribute modified code in the main open source licenses; public domain and shrink-wrap license included for comparative purposes

	In source code	In object code only (commercial use)
Shrink-wrap license	N/a	No
General Public License (GPL)	Yes (if GPL-ed)	No
Library (or Lesser) General Public License (LGPL)	Yes (if LGPL-ed)	No, unless linked code
Mozilla Public License (MPL)	Yes (if MPL-ed)	No, unless separate files (`larger work')
BSD license	Yes	Yes
Public domain	Yes	Yes

Yes = rights granted; no = rights not granted; n/a = not applicable.
All open source licenses stipulate that copyright remains with the original developer, and a copyright notice is to be retained in all modified versions.