IPR, Law and FLOSS: Building a Protected Common

Rishab Aiyer Ghosh†
United Nations University/Maastricht Economic Research Institute on Innovation and Technology (UNU-MERIT),
Keizer Karelplein 19, 6211 TC Maastricht, The Netherlands

Received 13 November 2006

This paper provides an overview of copyright and patents as they apply to software, and how open source depends on and uses some aspects of the IPR system for its existence, but may be threatened by others. It examines the incentives to release software as open source under different legal instruments, and compares impact of legal frameworks for open source on innovation to traditional frameworks such as patenting.

Keywords: FLOSS community, license, copyright, software patents, legal framework, intellectual property rights

In the case of what is one of the most economically productive online communities, and the focus of this text, Free/Libre/Open Source Software (FLOSS) developers, there is sometimes a misconception that laws are ignored or that the community’s efforts are ‘shared’ as public domain and thus ignorant of IPR concepts.

In fact, FLOSS communities are among the most formalized in cyberspace, with the basis of their functioning guided by licences under which their output is distributed, based on copyright law as a foundation.

The interaction between FLOSS communities and rights is a complex interaction between the actors (developers and other community contributors), artefacts (code and documentation) and legal frameworks as they determine the scope of intellectual property rights. To elaborate on this interaction, it is useful first to examine the way in which rights are treated by the current legal framework, in particular, copyrights and patents. Normally, once a work is created, it is exclusively appropriated by the creator, with a limited, temporary monopoly granted by the state. This monopoly provides the creator with the sole right to control access to the work, and the form it takes depends on the type of legal instrument involved. With copyright, the monopoly is over the reproduction of the work. It prevents follow-on creation by others without the permission of creator.

With patents, the work itself is not necessarily protected, as patents are granted on an invention, which may be abstract and may be implemented in different works. Thus, a patent can be granted to a creator (or inventor) without a work having been created. The monopoly provided to the inventor by a patent is different from that provided by copyright – it prevents follow-on creation by others without the permission of the inventor. But it also prevents others from independently creating implementations of the same or similar abstraction covered by the patent, without the inventor’s permission. Either way, the monopoly is what rewards the creator/inventor and provides an incentive for future creation.

With FLOSS, this monopoly for the creator, providing rights to the created artefact, is not used as an incentive to create. Incentive structures in FLOSS communities are more closely aligned to sharing of output rather than its appropriation. This introduces several complexities in the interpretation of who the creator is, and how (and by whom) rights are exercised.

Rights Claimed by FLOSS Developers

First, it is necessary to clarify that FLOSS developers do claim and exercise rights over their creations, even if this is done through unconventional uses of the legal framework. FLOSS artefacts – software, documentation – are not public domain, in the legal sense of the term, though they may be public goods in the economic sense. FLOSS refers to software to which the ‘Four Freedoms’ adhere (Stallman): users have the freedom to use, freedom to study, freedom to modify and freedom to share this software.

†Email: rishab@dxm.org
However, software authors own their code – under the Berne Convention, all copyrightable works are automatically covered by the copyright of the original creator at the moment of creation. No registration or notice – not even a copyright notice attached to the work – is required. Since software authors own their code, they are free to sell it, or indeed to ‘give it away’. They must do this explicitly, and can impose conditions. Conditions may, of course, perpetuate the ‘Four Freedoms’.

Although, the community has evolved its own implicit and explicit, informal rules and norms, legal foundation of the FLOSS community structure is in copyright law. Authors have the sole right to license their software to others, and software users must follow licence terms – otherwise they are infringing authors' copyright.

While software authors can safely ‘give it away’, this would literally be releasing software into the public domain and disclaiming all future rights to it. This is rare (and not even possible in some legal frameworks, e.g. in jurisdictions which provide for inalienable moral rights of the author). Instead, licences for FLOSS follow two broad models: permissive and reciprocal, and both involve the release to licensees of human-readable source code along in addition to machine-readable object code.

**Models of FLOSS Licensing**

The permissive licensing model is fairly close to public domain. It allows licensees broad rights to use, study, modify, distribute the software with few if any conditions. Most conditions relate to disclaimer of warranty issues. Examples of such licences include the Berkeley BSD licence, under which the popular operating system FreeBSD and its relations are distributed; the Apache Licence used for the market leader in web server software, Apache; and the MIT licence used for the X Window system of graphical user interfaces under Unix-like operating systems. As the names of some of these licences indicate, they originated in universities and are often referred to as academic licences.

The other licensing model, accounting for a majority of FLOSS projects is reciprocal. Quite different from the public domain, this model forms a ‘protected commons’. Licensees have broad rights to use and study the software. If they distribute the software, they must provide recipients access to the source code (providing them the freedom to study). They must also provide recipients with the software under the same terms, allowing recipients to further use, modify or distribute it. Licensees can only modify the software if the modified software is also distributed under the same terms, ensuring that recipients of such a derived work can further modify it. This ensures reciprocity by forming a ‘protected commons’ – authors are contributing their software into a commons with certain freedoms attached, and further modifications must be made available with the same freedoms protected.

The best known reciprocal licence is also the most widely used FLOSS licence, accounting for over 66% of FLOSS software projects (Freshmeat 2005), the GNU General Public License (GPL), with a further 6% distributed under the closely related Lesser GPL. The GPL is the licence used by the Linux kernel and several other large software packages. Other widely used reciprocal licences include the Mozilla Public Licence, used for the popular web browser Firefox; the Lesser GPL, used by OpenOffice, the main competitor to the Microsoft Office productivity suite; and the Common Development and Distribution License used by Sun Microsystems for OpenSolaris, the open source version of its respected server operating system.

It is interesting to note that this appreciation for requiring reciprocity when making a contribution to the public good is not limited to software. Creative Commons (CC), a foundation started by law professor, Lawrence Lessig, builds modular template licences for various types of content, such as text, art and music. These are intended to make it easier for rights holders to release their works to a commons, but under a broader set of rules than exist for free software. For example, CC licences can require attribution, or prevent commercial use – neither restriction is allowed by the open source or free software definitions. However, one of the attributes that can be asserted on a CC licence is ‘share-alike’ – a requirement that derived works be distributed under the same licence. This is the same enforced reciprocity as exists in software in the GPL, MPL, CDDL and other ‘copyleft’ licences. From approximately 140 million works licensed in June 2006 under CC licences, about 45% require ‘share-alike’ or enforced reciprocity. A further 24% require ‘no-derivatives’ (i.e. free to use, not to modify). Thus, among creators who release their works under CC licences for distribution and allow modification (derivatives), about 59% require reciprocity.
Conditions and Reciprocity

It should be noted that the ‘reciprocity’ obligation sounds and is often misinterpreted as stronger than it really is. The ‘commons’ is not formalized, and there is no obligation that licensees who modify software to make derived works ‘give back to the commons’ in a formal sense i.e. modified software does not need to be given away at no cost; nor does source code need to be published or provided to the original author. Indeed, such requirements would disqualify a licence from being a free software (or open source) licence. Reciprocal licences such as the GPL require that ‘recipients of software’ have the four freedoms; they do not require that the public at large have these freedoms.

The GPL, for instance, allows an author of a derived work to sell the work for 5,000 Euro a copy in only binary form (machine readable object code). However, all those who buy this software must, according to the GPL, be given the four freedoms. In particular, they must have the right to study the code, which is why the GPL requires that ‘recipients of object code’—in this example, the buyers—be provided with the source code at no significant extra charge. Similarly, the recipients have the right to modify and distribute the code with no further conditions; since they may distribute the code they received at no charge, if they so wish, or sell it for a lower price than they paid for it, charging high prices for the code alone, while allowed by FLOSS licences, is unsustainable under normal market conditions.

There are some examples of firms that actually follow a business model of charging significant fees for free software. Ryxeo is a French firm that was producing server software customised for schools. Schools were their main clients. The software they used was the so-called LAMP tools (Linux, Apache, MySQL and Perl—all free software) along with other pre-existing free software which they customized, and to which they added their own code. As their software was derived from other GPL packages they had to license it under GPL. They sold the software with the hardware and support services as a package. While they initially published the software, later versions were no longer distributed but only sold to their customers as part of a total support package. Their customers, the schools, did not have to disclose the source code, let alone further distribute the software—although, as they received it under the GPL, they could have done this. As a result, the price was not driven down to zero, and the business model was sustainable. In a different environment, it is feasible that customers of free software may not want to publish or further distribute what they buy so as not to provide an advantage to their competitors (perhaps not an issue of particular concern for schools).

This case shows that in specialized markets with a high degree of customisation involved, a software-purchase business model may work even with reciprocal FLOSS licences. It emphasizes the distinction between a commons for users of software—who may have to pay for it to first get hold of the software—and a commons for everyone. FLOSS is the former, though in practice it often appears like the latter.

The reciprocal conditions imposed by FLOSS licences such as the GPL are unusual, though they have since been widely reproduced (including, as described above, in Creative Commons licences for non-software works such as art and text). Several commentators, including most famously Microsoft Senior Vice President (now Chief Research and Strategy Officer) Craig Mundie have raised questions as to the validity of the GPL’s reciprocity requirements, going so far as to claim that reciprocal conditions expropriate the intellectual property rights of the authors of derivative works. Such claims are astonishing, since authors of derivative works have no particular right to create them in the first place. Copyright law prevents anyone from modifying or distributing software without the explicit permission of the copyright holder—permission granted usually through a licence. The copyright holder, choosing to grant the permission to modify, is free to set any conditions on the licence. It is noteworthy that most FLOSS licences take the form of a permission grant, which is not a contract (requiring agreement) under several common law jurisdictions. This is because they are explicitly designed to grant permission to licensees to perform actions that copyright law prevents them from performing—such as, copying, modifying or distributing. Licensors may provide such permission under any conditions—such as, the payment of royalties, or cross licensing of derivative works. Such conditions may be found in commercial licences. In the case of reciprocal FLOSS licences, permission for creating derivative works is granted on the condition that such works are distributed under the
same licence.

It is worth comparing restrictions in FLOSS licences to licence terms commonly found in proprietary software licences. FLOSS licences are designed only to allow actions that are prevented by copyright law – thus, licence conditions do not restrict what is allowed, but place conditions under actions normally forbidden are allowed. Proprietary licences may place conditions even on activities that are normally permitted by copyright law – thus, they are necessarily contracts, requiring the agreement of the user.

For example, a FLOSS licence gives you permission to sell copies of the licensor’s software, which would normally be forbidden under copyright law. It may place conditions on this permission. But a FLOSS licence normally cannot place conditions on your ability to copy parts of software, make a personal backup copy, or other activities allowed by copyright law under ‘fair use’ or equivalent terms.

In contrast, proprietary software licences often place several conditions on software use that go beyond copyright law – they prevent you from doing what copyright law allows. This can include restrictions on the type of use of the software, type of copying, or even conditions completely unrelated to copyright. A notorious case is perhaps that of Network Associates’ McAfee VirusScan software, whose licensing terms, preventing users from publishing reviews of the product, led to the Attorney General of New York suing the company in 2001.

Without following licensing conditions, users who distribute a work or make derived works are making unauthorized copies, thus infringing copyright. Indeed, in a rare court case concerning the distribution of modified versions of netfilter/iptables, a tool in Linux, a German appeals court ruled that even though the GPL itself may not entirely be valid in German contract law, it was the only thing that granted permission to the accused to distribute the software. Thus, the terms had to be obeyed, otherwise it was a simple case of copyright infringement.

**Reciprocity and Collaboration**

Creators of a number of open source projects, with the aim of maximizing use, have chosen permissive rather than reciprocal licences. Some of the early choices have been without much discussion, almost by default. The prototypical permissive licence is the Berkeley Software Distribution (BSD) licence, used for the various versions of BSD Unix. This enormously influential systems software suite has, in great part due to its licence, provided the basis for the original Internet code for Microsoft Windows and the underlying operating system for all Apple Macintosh computers since 2002, several parts of the Linux kernel as well as spawning several open source variants (FreeBSD, OpenBSD, NetBSD). Originally copyright the Regents of the University of California, the licence was typical of the ‘academic’ publication ethic.

Similarly, brief and permissive (and academic in origin), the MIT license or X license originated to distribute the X Window System, the graphical user interface (GUI) for Unix that originated in 1984 at MIT and is now the basis for most GUIs running on versions of Unix, Linux and BSD.

Both these licences were implemented before the first version of the GPL, thus before the notion of reciprocal licensing became widely known. The GPL’s legal innovation was truly remarkable, and the most significant permissive licence that followed was probably the Apache licence in 1995. This was written for the Apache web server, an open source application written not by academics but by Internet professionals and website administrators. The GPL was already dominant open source licence and the discussion among the Apache developers, about whether or not to require reciprocity, is something many subsequent projects have faced, with varying degrees of argument. Apache chose to maximize its user base, and to encourage contributions to the commons through gentle social pressure rather than binding restrictions. Indeed, Apache’s user base was maximized – it became the most used web server within a year of its release, and has held a steady two-thirds of the total web server market since 2000.

Some of the scripting languages and content management systems, tools used (among other things) to make websites interactive have also used permissive licences. But the Linux kernel and the majority of open source software uses reciprocal licences. One reason is that reciprocal licences are drafted to enforce reciprocity through ‘recursion’ – typically, a derived work must be distributed under the same licence. Thus, new software that reuses old GPL software – code reuse being one of the hallmarks of open source development – must be licensed as GPL. For those not strongly opposed to reciprocal licensing, choosing the GPL is a fair trade for getting
access to an ever huger codebase to reuse. Thus each reciprocal licence is automatically designed to be dominant, and the most popular, or oldest, will by default dominate the entire licence space.

However, one cannot say that most developers are against reciprocal licensing, or even neutral towards it. For rational actors, reciprocal licences may be a better choice than a permissive licence. Certainly, one feature of ‘giving your work away’ that is hard to justify in a narrow-minded rational actor approach is the threat of competitors benefitting what you give away, or more generally, the threat of free-riding. This occurs less with reciprocal licences, since competitors can benefit, but they cannot exclusively appropriate the benefits. They can share, but not steal. If they adapt or improve the work, they must return it to the commons, allowing the original creator to benefit from the improvements. Reciprocity ensures that development remains collaborative, and cannot be exclusively appropriated.

Reciprocity and Incentives

And reciprocity provides incentive for new contributors, including firms. 60% of developers think\textsuperscript{20} the role of a licence is ‘To prevent others from appropriating the software we have created’ (FLOSS-US survey\textsuperscript{20}), thus showing that they are not altogether (if at all) altruist and may frequently be choosing reciprocal licences with the selfish motive of ensuring their access to future improvements.

The preference among developers for reciprocity is not limited to independent individuals. According to a survey of 146 Italian firms, firms that release open source software prefer to use the GPL because ‘it allows to keep the code open and forbids competitors to turn it into proprietary’.\textsuperscript{21}

This has even been a concern for the public sector. For example, in a study conducted to examine the possibility of the European Commission releasing a software application it owns under an open source licence, a key condition was that ‘the Commission requires protection against appropriation of application by third parties’.\textsuperscript{22} The recommendation, based on this requirement, was to use a licence with a reciprocity clause, i.e. a copyleft licence, such as the GPL.

Enforced reciprocity changes the incentives involved in deciding whether to distribute software (or other information) under proprietary protections, or as open source. Instead of a binary choice between proprietary and public, implied by commentators such as Lerner and Tirole (2002)\textsuperscript{23}, contributors in fact face a more sophisticated choice, as illustrated in Table 1. Commentators have often assumed a choice limited to the contributor’s own appropriation of a work, with the corollary assumption that a competitor B can benefit if the contributor A does not exclusively appropriate it.

<table>
<thead>
<tr>
<th>Exclusive appropriation by others (competitors)</th>
<th>No appropriation by competitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive appropriation by contributor A</td>
<td>Proprietary</td>
</tr>
<tr>
<td>No appropriation by contributor A</td>
<td>Permissive Reciprocal licence</td>
</tr>
</tbody>
</table>

In fact, the profit to a competitor B of access to a contributor’s work $P_{accessB}$ is quite different (and potentially much smaller) than the benefit to a competitor of exclusive appropriation of the contributor’s work, $P_{exclapprB}$. Similarly, the corresponding loss for the contributor of a competitor having access to its work $L_{accessA}$ is much lower than the loss to the contributor of having a competitor having the ability to exclusively appropriate (improvements to) the work, $L_{exclapprA}$.

\[
P_{accessB} < P_{exclapprB} \\
L_{accessA} < L_{exclapprA}
\]

In the context of non-rival information goods, free riding applies more to the case of others’ exclusive appropriation than to simply the access by others of a contributor’s work. The rational choice for a potential contributor is to minimize its losses and if possible minimize the benefits to its competition. Thus, once a decision has been made by firm A not to exclusively appropriate a work, it is likely to choose a form of distribution that enforces reciprocity. Firm A will choose to lose $L_{accessA}$ and concede competitors a profit $P_{accessB}$ with a reciprocal licence. A permissive licence allows exclusive appropriation by competitors of A’s work, and this is in addition to basic access to A’s work. So, a permissive licence would lead to a greater loss ($L_{accessA} + L_{exclapprA}$) and a greater win for the competition ($P_{accessB} + P_{exclapprB}$).

Choosing to release a work as open source in the first place is discussed in later chapters. While such a
choice does require some incentive to offset the loss of a contributor to giving up exclusive rights to a work, the incentive required is lower if it is to compensate only for $L_{access}A$, rather than for $L_{access}A + L_{exclusive}A$.

Indeed, as there are several inherent benefits in providing competitors access to a work, it may be fairly easy to compensate for $L_{access}(A)$. Inherent benefits to using permissive licences and allowing competitors to exclusively appropriate benefits are fewer.\(^{25}\) This may explain why permissive licences are typically chosen for historical reasons, or when a community already exists around a work that is being extended, and can be relied on to apply some form of social pressure to limit free riding (exclusive appropriation without contributing in return). Such communities can be informal, as with the Apache developer community (many of whom are members of the Apache foundation). Or, they can be formal communities, as with the Object Web or Eclipse communities, both formed as consortiums of firms that contribute much of the work.

**Reciprocity as a Framework for Disclosure**

As discussed previously, patents, which are justified on the basis of promoting disclosure (and therefore follow-on innovation), are not really succeeding at that task.

Arora *et al*\(^{26}\) found that ‘patent disclosures appeared to have no measurable impact on information flows from other firms, and therefore no measurable effect on R&D productivity’. Arundel\(^{27}\) found that ‘a consistent result in survey research on the use of patent databases is that they are among the least important external information sources available to firms’. His analysis of 12445 firms’ responses to the CIS survey results\(^{28}\) shows that between 5% and 18% of small and medium-sized firms find patents to be a useful source of information\(^{29}\).

In the case of software, surveys show\(^{30}\) that more firms think free software source code is an important source of new ideas (17%) than patent databases (5%). The opinion of individual innovators (engineers) is perhaps more relevant as questionnaires on patents sent to firms are likely to be answered by the legal department than by innovators. Far more innovators within firms\(^{31}\) think source code (41%) or journal publications (68%) are moderately or very important sources of new ideas, than patents (24%).

While we do not know how much of this software that is source of new ideas is licensed under reciprocal terms, these data show that open source software is succeeding in providing disclosure, while patents are not. This is certainly at least in part due to reciprocal licensing, which provides a legal requirement to disclose (much as patents are supposed to do). Without reciprocal licensing, disclosure would be only due to social, economic or other incentives, but not a requirement, and would presumably be reduced.\(^{32}\)

If a legal framework is required to promote disclosure and follow-on innovation, there is, therefore, some evidence to justify an argument that reciprocal open source licensing provides a more effective framework than the current patent regime.

**References**

1. With no claim of (copy) right, works in the public domain can be used in any way by any one; Samuel Edward ‘The Public Domain In Copyright Law by Edward Samuels’, *Journal of the Copyright Society*, 41 (1993) 137.
6. http://wiki.creativecommons.org/License_statistics—note that the licence break-up and the total figure of 140 million are not necessarily based on the same sample of works, as the break-up uses Yahoo’s search engine and the total figure is calculated using Google’s search engine.
9. In many European countries, there is a right to make a ‘private copy’.


15 http://www.opensource.org/licenses/mit-license.php.


17 http://www.apache.org/licenses/LICENSE-2.0.


23 ‘Why should top-notch programmers contribute freely to the provision of a public good’?

24 As discussed in later chapters; for firms, such inherent benefits include spreading the cost of maintenance and improvement across other firms who use and (possibly due to reciprocity requirements) contribute to a work, as well as spreading the cost of marketing across users.

25 Limited mainly to spreading the costs of marketing across users-permissive licences are often justified as a way to ensure the broadest possible user base, as in the case of the Apache web server.


29 The share is 34% for large firms, but even they find patents less useful than other sources of information, such as customers, suppliers, conferences and journals, trade fairs, and competitors.


31 Arundel et al 2006, (supra note Error! Bookmark not defined.) shows consolidated data for all respondents; figures included here are for individual innovators employed at private companies, i.e. excluding those employed at public organizations or research institutes.

32 Several firms embrace disclosure for other incentives, e.g. when they contribute to Apache software which has no reciprocity requirements. However, several firms try to evade the disclosure requirements of reciprocal licences such as the GPL, when the GPL’s legal requirement to disclose provides a useful mechanism. The court case referred to in supra note Error! Bookmark not defined. is one example, and the GPL Violations Project (http://gpl-violations.org/) contains many others.