International Free and Open Source Software Law Review

Volume 02} Issue 01} June 2010

Contents

ISSN 977-18776920-1 03
9 771877 692001

Available online at: http://www.ifosslr.org

Volume 02} Issue 01} June 2010

About IFOSS L. Rev.

The Black Duck Legal Webinar Series

Watch live and recorded webinars covering basic and advanced topics on open source software issues.

www.blackducksoftware.com/legalwebinars

Visit to see a list of on-demand webinars and dates for new upcoming topics.

blackduck tm

www.blackducksoftware.com

openinventionnetwork SM

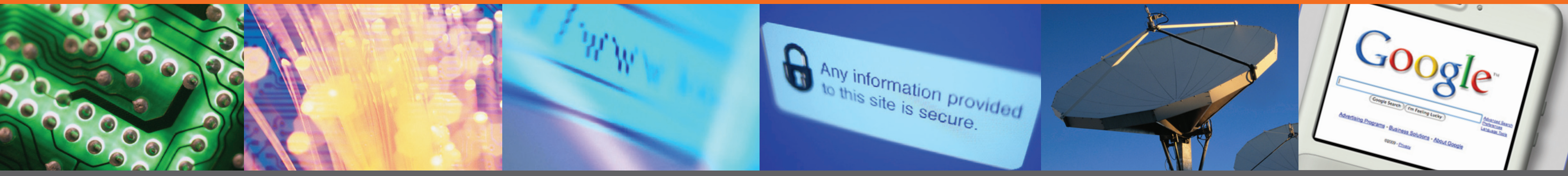
A Distinct Mission: Keeping Open Source Open

Open Invention Network SM is a collaborative enterprise that enables innovation in open source and an increasingly vibrant ecosystem around Linux by acquiring and licensing patents, influencing behaviors and policy and protecting the integrity of the ecosystem through strategic programs such as Linux Defenders. It enables the growth and continuation of open source software by fostering a healthy Linux ecosystem of investors, vendors, developers and users.

Open Invention Network (OIN) has considerable industry backing. It was launched in 2005, and has received investments from IBM, NEC, Novell, Philips, Red Hat and Sony.

One of the key methods in which OIN promotes Linux is by acquiring patents across a wide range of technologies.

Patents owned by OIN are available royalty-free to any company, institution or individual that agrees not to assert its patents against the Linux Community.



Linux Defenders (www.linuxdefenders.org)

Open Invention Network is seeking creative and energetic individuals from the Open Source Community to actively contribute to Linux Defenders.

Linux Defenders offers the Linux and broader open source community a unique opportunity to harness its collaborative passion, intelligence and ingenuity to ensure Linux's natural path of growth and innovation. The free program is designed to benefit open source innovation by significantly reducing the number of poor-quality patents that might otherwise be used by patent trolls, or other organizations, whose behaviors and business models are in opposition to those of the open source community.

The success of the program will be driven by contributions from the open source community. OIN encourages prior art and raw invention contributions that will be leveraged with the governmental patent authorities to invalidate poor-quality patents and patent applications.

openinventionnetwork SM
Research Triangle Park Center
4819 Emperor Boulevard
Suite 400
Durham, NC USA 27703
T: +1 919 313 4904
F: +1 919 313 4905
info@openinventionnetwork.com
www.openinventionnetwork.com

Policies and bibliographic information

Copyright and licensing statement

IFOSS L. Rev. is committed to the improvement of understanding of legal issues in digital society. A licensing statement is therefore attached to each article, clearly outlining the particular terms which apply to the article. Most use Creative Commons licences with special exceptions for translations.

Graphic design

The Editorial Committee wishes to thank Tomasz Politański Design for its logo and associated graphic design work.
http://tomaszpolitanski.com

Publisher & sponsorship

IFOSS L. Rev. is published by its Editorial Committee, with financial and administrative assistance from NLnet Foundation and Mozilla Foundation. Please note that neither NLnet Foundation nor Mozilla Foundation accept correspondence on behalf of this publication. All correspondence should be directed to the Editorial Committee via email (see below). The Open Invention Network also sponsors this issue of IFOSS L. Rev.

Editorial policies

IFOSS L. Rev. accepts articles for publication from qualified personnel based on the criteria available to view on its web site (http://www.ifosslr.org). Submissions are welcome from all, and authors are strongly encouraged to read the style and content guidelines available on the web site. The review operates an anonymous peer review system for articles as appropriate, and expects all authors to meet the highest standards of scholarship and integrity.

Bibliographic information

The authors explicitly encourage libraries, archives and educational institutions to hold copies of IFOSS L. Rev. in their collections, in electronic and/or printed form. All users are advised that articles may occasionally be updated after publication. Linking back to original copies on the IFOSSL. Rev. web site, where authoritative versions are archived, is strongly recommended. Please contact the Editorial Coordinators for further information on best practices. It partipates in the CrossRef system.

ISSN: 1877-6922

Publication schedule

IFOSS L. Rev. is published biannually. Submissions for publication are welcome at any time, but publication deadlines exist for each issue. For the latest information on papers sought and deadlines for submission, please consult the IFOSS L. Rev. website or contact the Editorial Coordinators at (admin@ifosslr.org)

As the world's largest IT company, HP simplifies technology experiences for all customers – from individual consumers to large enterprises. HP helps businesses focus on innovation rather than IT operations by offering robust solutions built on HP inventions, open source software, and industry-standard hardware delivered as a converged infrastructure. In fact, as the leading provider of scalable Linux platforms, HP ships one Linux server every minute.

HP not only uses open source but also contributes expertise to hundreds of open source projects worldwide. In fact, we built tools for open source governance, then released them as an open source project called FOSSology, and now lead the thriving FOSSBazaar community with best practices.

Outcomes that matter.

Contact an HP representative today to learn more about our open source & Linux expertise, platforms, and services.

www.hp.com/go/linux
www.hp.com/go/opensource

Licence Profile: BSD

Andrew Sinclair[a]

(a) Legal Counsel, Canonical USA, Inc.

DOI: 10.5033/ifosslr.v2i1.28

Abstract
The BSD software licence is one of the most popular open source software licences, with simple permissive licence terms. This article is a short overview of the licence, examining its elements and their interpretation.

Keywords
BSD licence; Law; information technology; Free and Open Source Software

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

The BSD licence is the flagship representation of the "non-copyleft" open source licensing model. Its terms are unquestionably simple when compared to many other open source licences, yet the BSD licence carries great significance. When measuring popularity by frequency of use, the BSD licence consistently ranks at the top of the list after the GPL family of licences.[1] This makes the BSD licence the most common non-copyleft licence, and in holding this status, the BSD licence is often the first example cited when comparing copyleft and non-copyleft licensing models.

When viewed as the primary representation of a major open source licensing model, the BSD licence's language is marvellously simple, and perhaps this is because the simplicity of the model demands a simple embodiment. There are similar licences, like the MIT licence and the historical permission notice, which also use very few words to represent the non-copyleft model. The BSD licence's few words, however, require some interpretation to fully cover the rights and obligations that open source communities have come to associate with it.

1 *See* http://sourceforge.net/softwaremap/? (showing the number projects hosted on SourceForge.net using various licences, with BSD in the third position after GPL and LGPL); *See* http://www.blackducksoftware.com/oss/licenses#top20 (showing the frequency of licences appearing in Black Duck Software, Inc.'s database of open source software, with BSD in the fourth position after GPL, LGPL, and the Perl Artistic licence)

Parsing the licence

The BSD licence has a three-part structure. It sets forth a basic copyright notice, has a short licence grant, and has a warranty disclaimer and limitation of liability clause.

Copyright notice

The BSD licence's copyright notice follows the style of a traditional proprietary copyright notice. It sets out the author's name and the date of the work consistent with the US Copyright Act.[2] When the United States joined the Berne Convention in 1988, it revised its Copyright Act to eliminate the notice requirement.[3] However, copyright notices are still extremely common, and they serve still serve the practical purpose of identifying the copyright owner to recipients of the work. The copyright notice in the BSD licence also makes sense given the timing of the BSD licence's first use. The original version of the BSD licence was first used in 1980 in connection with the Berkeley Software Distribution. As this was well before the new US law removing the notice requirement became effective, the notice would have been required for enforceability under US law.[4]

The second part of the BSD licence's copyright notice is the familiar "all rights reserved" notice, which seems to contrast the broad set of rights granted by the rest of the licence. Surely, not all rights are reserved, as the author is granting many rights in the same instrument as the notice (the BSD licence), but it is an interesting relic of the more reserved closed source licensing model where such a notice would likely be followed by much more narrow licence grant. None of the other common open sources licences include or suggest including "all rights reserved" in the copyright notice or anywhere in the licence.[5]

The licence grant

The heart of the BSD licence is its one-sentence licence grant and short list of conditions: "Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met . . ."[6] The only right explicitly granted is the right to distribute, but there is a strong suggestion that a right to modify or prepare derivative works is also present. The "source and binary forms" language suggests that the source code version may be available, which would have little practical use if the recipient does not also have a right to modify it. Furthermore, "with or without modification", while not explicitly granting the licensee a right to modify, has no other plausible interpretation; the right to distribute "with or without modification" presumes that someone has the right to modify. If this referred to the licensor's right to modify,

2 United States Copyright Act, 17 U.S.C. § 401 (2009)

3 Berne Convention Implementation Act of 1988, Pub. L. No. 100-568, 102 Stat. 2853, enacted October 31, 1988

4 Deek, Fadi P. & McHugh, James A. 2008 *Open Source: Technology and Policy.* New York: Cambridge University Press, p. 337

5 *E.g.* http://www.gnu.org/licenses/gpl-2.0.html, http://www.gnu.org/licenses/gpl.html, http://www.apache.org/licenses/LICENSE-2.0.html, http://www.opensource.org/licenses/mit-license.php

6 http://www.opensource.org/licenses/bsd-license.php

there would be no need to express this right; whether software version is modified by the licensor prior to licensing would have no effect on granting a licensee a right to distribute.

It is clear that the licensee has a right to distribute the work, and it would be hard to argue that the licensee does not also have a right to modify. However, one of the most significant rights under copyright law is entirely missing from this grant: the right to reproduce the work. Some right of reproduction could be read into the right to modify, as the type of work the licence covers is computer code, and it is impractical to suggest that the licensee may modify and distribute a computer software work but may *not* reproduce that software. The expressly granted right to use could bolster this position; with respect to software, use often requires some form of reproduction. A second and perhaps stronger solution to the omission of the right to reproduce the work is to look beyond the strict legal interpretation and consider the intent of the licensor. The fact that the licence includes the superfluous "all rights reserved" is not helpful in construing the licence to grant a right that is not explicitly granted, but the open source community has treated the BSD licence as permitting a right to copy.[7] With decades of use assuming this right, this convention cannot be ignored.

Licence conditions

The "new" or "3-clause" version of the BSD licence contains three conditions:

> *"* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.*
>
> ** Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*
>
> ** Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission."*[8]

The first condition is relatively simple, and it is stated very simply in the licence. It is also a condition that is extremely easy to satisfy, as failure to retain a notice would require the act of removal. However, the second condition may be one that is frequently overlooked. When a licensee compiles the source code into binary form and distributes that binary, the second condition would require the licensee to *add* a copy of the licence to the binary's accompanying documentation or other other materials. This isn't entirely consistent with the common view that the BSD licence only requires "credit" or "attribution". Attribution is required in the form of the copyright notice portion of the licence, but merely attributing the work to a particular author would

7 Meeker, Heather J. 2008 *The Open Source Alternative: Understanding Risks and Leveraging Opportunities.* Hoboken(NJ): John Wiley & Sons, Inc. p. 28 (Noting that "these rights are universally understood to be granted under this license."), DiBona, Chris, Ockman, Sam & Stone, Mark eds. 1999 *Open sources: Voices from the Open Source Revolution.* Sebastopol(CA): O'Reilly & Associates, Inc. p. 164 (Brian Behlendorf writes of BSD-style licences: "[By] and large it can be summed up as, ' Here's the code, do what you like with it, we don't care, just give us credit if you try and sell it.'")

8 http://www.opensource.org/licenses/bsd-license.php

not satisfy the condition. It should still be relatively easy to comply with this condition. In the context of distributing BSD licensed software as open source, the inclusion of the source code version would likely satisfy the requirement. If the licensee complied with the first condition by not removing any notices or licence information, and if adequate notices were already present in such source code version, the source code version would constitute "other materials" distributed with the binary version which includes notices adequate to satisfy the second condition. Complying with the second clause in a proprietary context requires more care. The licensee who redistributes the binary must add a copy of the licence to documentation or other materials.

The third condition is a prohibition on using certain names to promote a product, but this does not seem to alter the rights of the licensee. In most jurisdictions, trademark law already prohibits the kind of unlicensed endorsement addressed by this condition. However, the condition is not meaningless; while it may be that a contributor or copyright owner would have a trademark infringement claim against a licensee who uses its name without permission, the condition ties such unauthorized use to the copyright licence. The licensor therefore has an additional remedy (a copyright claim) available should a licensee promote a product using the licensor's name without permission. The third condition may also serve the practical purpose of reminding licensees that they should not use the licensor's name for promotional purposes. Many readers of the BSD licence will not be lawyers versed in local trademark law, so the third condition's setting out the promotional restriction in plain English is helpful to licensees who may not otherwise be aware of this prohibition.

Warranty Disclaimer and Liability Exclusion

The final part of the BSD licence is its one-sentence disclaimer of warranties and one-sentence exclusion of liability. As software licensed under the BSD licence is done so without charge or royalty, it is appropriate that licensees do not receive commercial guarantees. Furthermore, the potentially ongoing distribution stream enabled by licences like the BSD licence would make warranties and liability terms difficult to implement. The BSD licence takes the distribution and re-licensing model into account in both the warranty disclaimer and the liability exclusion by applying these to all upstream copyright holders and contributors.

Compatibility

Advertising clause

The original version of the BSD licence included an additional condition: “All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the <organization>.”[9] In addition to the problem of the potential inconsistency between this condition and the condition prohibiting promotion or endorsement, Richard Stallman cited this condition as practically problematic.[10] If developers started adding code to the work, the list of required advertising notices would continue

9 http://www.gnu.org/philosophy/bsd.html
10 *Id.*

to grow until it became unmanageable.[11] The Free Software Foundation has also cited the advertising condition as triggering a conflict with the GPL.[12] In 1999, the University of California removed this condition of the BSD licence, and the version with the advertising restriction is not an approved licence by the Open Source Initiative.[13]

Other compatibility issues

While the BSD licence and similar highly permissive licences are generally thought to be compatible with copyleft licences like the GPL, the legal effect of combining code under the BSD licence with code under a copyleft licence is not always clear.[14] The BSD licence does not include an express right to sublicense, so if the BSD licence is compatible because the code it governs is "re-licensed" under the copyleft licence, the licensee must rely on the licensor's intent and community interpretation to read this sublicense right into the BSD licence's terms. However, the typical open source model is a direct grant from the copyright owner to the licensee, not a sublicence.[15] If, instead of a sublicence, the BSD licensed code is combined with the copyleft code but continues to be licensed under the BSD licence, this would seem to conflict with the terms of the copyleft licence, which will typically require that derivative works are licensed under the copyleft licence. Resolving this apparent conflict in the legal context would require analysis of the applicable copyleft licence and application of the particular facts and circumstances. However, it is once again helpful to consider the community interpretation of the BSD licence and copyleft licences, which generally considers the BSD licence to be compatible with copyleft licences.[16]

Conclusion

The BSD licence is significant due to its popularity and the simple non-copyleft licensing model it represents. In a few ways, the BSD licence lacks clarity as a legal document, as it does not include some express licence grants that are otherwise reserved under copyright law. However, the BSD licence's long history of use and shared community interpretation help to resolve the apparent conflict between a strict textual interpretation and the licence's practical use. The BSD licence's language also includes some clues as to rights that are assumed, which further support the view that it is indeed a very permissive licence. BSD licence compliance is relatively straightforward, but a licensee who has a an over-simplistic understanding of the BSD licence may find it too easy to overlook the requirement to add notices to documents when distributing BSD licensed software in binary form. Overall, the BSD licence is a simple licence, but not quite as simple as a one-

11 St. Laurent, Andrew M. 2004 *Understanding Open Source & Free Software Licensing*. Sebastopol(CA): O'Reilly Media, Inc. p16 (noting that the requirement to including references to numerous preceding works can become a burden)

12 http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses

13 Williams, Sam, 2002 *Free as in Freedom: Richard Stallman's Crusade for Free Software.* Sebastopol(CA): O'Reilly & Associates, Inc. p. 140, http://www.opensource.org/licenses/bsd-license.php

14 *See e.g.* http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses (listing licences that the Free Software Foundation considers compatible with the GPL)

15 Meeker, Heather J. 2008 *The Open Source Alternative: Understanding Risks and Leveraging Opportunities.* Hoboken(NJ): John Wiley & Sons, Inc. p. 29, *see* http://www.gnu.org/licenses/gpl.html (GPLv3 says, "Sublicensing is not allowed; section 10 makes it unnecessary.")

16 *See e.g.* http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses (showing that the Free Software Foundation considers the BSD to be compatible with the GPL)

sentence “do whatever you want, but include the licence terms” summary would reveal.

About the authors

Andrew Sinclair *is Legal Counsel at Canonical, the commercial sponsor of Ubuntu.*

GNU GPL 2.0 and 3.0: obligations to include licence text, and provide source code

Neil Brown[a]

(a) Solicitor, Privacy, Security and Content Standards, Vodafone Group

DOI: 10.5033/ifosslr.v2i1.31

Abstract
An exploration as to whether, under the terms of the GNU GPL 2.0 and 3.0, a distributor of covered code is entitled to (a) provide a copy of the relevant licence in electronic form, or else via a link to an online location; (b) rely on online distribution of source code without the need to offer source code on a physical medium; and (c) rely on a third party's distribution of the corresponding source code.

Keywords
GNU GPL 2.0, GNU GPL 3.0, compliance

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

Following on from his presentation at the Free Software Foundation Europe European Legal Network annual conference in Amsterdam, Neil Brown explores three common compliance topics relating to the GNU GPL 2.0 and GNU GPL 3.0, through an analysis of each of the licences, and the environments in which they were created.

These topics are:

- Obligations of a distributor to distribute the text of a licence along with the covered code;
- Whether a distributor is entitled to rely on online provision of source code, without the need to offer source code on a physical medium; and
- Whether a distributor can rely on a third party's distribution of the corresponding source code of a covered work.

Obligations of a distributor to distribute a copy of the licence text along with the covered code

Whether through attempts to lower their environmental impact, to improve the out-of-the-box experience, or otherwise, many companies are increasingly keen to minimise the volume of paperwork provided with each of their products. However, what steps can a company take in respect of the text of the licences of the GNU GPL 2.0 and 3.0?

s1, GNU GPL 2.0[1] provides:

> *"You may copy and distribute verbatim copies of the Program's source code … provided that you … give any other recipients of the Program a copy of this License along with the Program."*

s4, GNU GPL 3.0[2] provides:

> *"You may convey verbatim copies of the Program's source code … provided that you … and give all recipients a copy of this License along with the Program."*

Although the scope of the obligation to provide the licence text differs slightly,[3] the obligation on the distributor[4] is clear; the distributor is obliged to give a copy of the relevant licence along with the Program.

However, there is no requirement for the copy of the licence to be in physical form; the text simply refers to "a copy", and, on this basis, an electronic copy of the relevant licence is sufficient. Indeed, a licence which required otherwise would be highly impractical in an environment in which software is distributed so widely in electronic format – to require a physical copy of the licence text would be to require a letter or a fax to be sent to each person who downloaded covered code. Electronic distribution could take a number of forms, including incorporation within the interface of a device, which is accessible from another device (e.g. via the "About" section of a router, which has a web interface), storing the licence on, and making it accessible via, the product in question (e.g. an electronic file stored on a mobile phone or a media player, which can be opened by the user on the device itself), or by inclusion on optical media supplied with a product.

Whilst neither licence places a restriction on the format of an electronic copy of the licence, it would be good practice to provide the licence in a format which is accessible to the recipient of the product, such that he is able to read the licence without needing to install non-standard software. For example, if a distributor were to include a copy of the licence on a mobile device, it would advisable to ensure that it could be viewed on that mobile device in an out-of-the-box state. Similarly, the use of restricted formats should be avoided; the Free Software Foundation provides plain text, and other unrestricted, copies of the licences which could be used in pursuit of this

1 http://www.gnu.org/licenses/gpl-2.0.html (All URIs in this article were verified on 14th April 2010)
2 http://www.gnu.org/licenses/gpl.html
3 For example, the GNU GPL 2.0 requires that a copy of the licence is provided to "any other recipients", whereas, under the GNU GPL 3.0, the licence text must be provided to "all recipients".
4 The term "distributor" is used throughout this article to refer to both those who distribute code under the GNU GPL 2.0, and those who convey code under the GNU GPL 3.0.

objective.

Although electronic distribution of licence text is not restricted by the terms of the GNU GPL 2.0 and 3.0, mere provision of a URL to an online copy of the licence is not a permissible alternative to a physical copy. Each of the licences requires that a copy of the licence be provided "along with the Program"; "along" implies concepts of togetherness and accompaniment, neither of which is satisfied by the inclusion of a URL for a website which a recipient must visit to procure the licence in question. The Preamble to each of the licences confirms this approach, providing that, when one is distributing covered code to another, one must "show them these terms so they know their rights"; merely enabling a user to access a copy of the licence on the Internet neither accompanies the covered code with the licence, nor shows that licence to the recipient of the code.

To conclude this section, then, it is clear that the distributor is required to provide a copy of the licence text along with the covered code, and that, whilst an electronic copy of the licence text is permissible, a mere link to an online copy of the licence is unsatisfactory for the purposes of the GNU GPL 2.0 and 3.0.

Is a distributor entitled to rely on online provision of source code without the need to offer the source code on a physical medium?

This section seeks to address two common questions in respect of obligations pertaining to source code. Firstly, whether making a copy of the corresponding source code[5] available online is sufficient to meet a distributor's requirements under each of the licences, and, secondly, whether it is permissible for a distributor to have obligations in respect of source code provision performed by a third party.

Making source code available online

In respect of offline distribution, s6, GNU GPL 3.0 provides that where object code is conveyed in a physical product, it must be either:

- accompanied by source code on a durable physical medium; or
- accompanied by a written offer either:
- to give the source code on a durable physical medium; or
- offering access to the source code from a network server.

In respect of online distribution, s6 further provides that where object code is distributed by offering access from a designated place, the distributor must offer equivalent access to the corresponding source code, or where distributed by peer-to-peer transmission, the distributor must inform other peers of the online location of the source code.

5 Rather than just the source code form of the object code, a distributor may be required to supply installation scripts and the like, to enable utilisation of the source code.

As such, a distributor of object code licensed under the GNU GPL 3.0 is never obliged to provide a copy of the corresponding source code on a physical medium, although it remains an option for the distributor to do so, in respect of a distribution of object code on a physical form[6]. However, a distributor is always entitled to rely on the distribution of the appropriate source code from a network server, provided that, where distribution of the object code is in a physical product[7], the distributor accompanies the object code with a qualifying written offer to make such provision.

Section 3 GNU GPL 2.0 provides that covered code can be distributed in object form, provided that the distribution is accompanied with the complete corresponding machine-readable source code on a medium customarily used for software interchange, or with a written offer to give any third party a copy of the corresponding source code on a medium customarily used for software interchange.

In respect of online distribution of object code, the last paragraph of s3 provides that:

> *"[i]f distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code."*

As such, where object code is offered online, offering "equivalent access" to the corresponding source code satisfies a distributor's obligation to distribute source code, even if the source code is not actually downloaded by the user downloading the object code.

However, in respect of offline distribution of object code, in order for online availability of source code to be permissible under the GNU GPL 2.0, such online availability must fall within the definition of being "give[n] … on a medium customarily used for software interchange". As such, one must consider whether making source code available for download by a third party is the same as "giving" that source code to the third party, requiring, as it does, an extra act by that third party to secure the source code. Similarly, "medium" infers a physical object, rather than via a protocol for online transmission. Thus, on a literal interpretation of the licence, online distribution of source code alone[8] is insufficient for offline distribution of object code.

Whilst such a conclusion may seem incongruous in the light of modern day Internet capability, it makes far more sense when one considers the state of Internet access at the time in which the licence was drafted; in 1991, access to the Internet was far less prevalent than it is today, which meant that, were mere online distribution acceptable, in reality, many recipients of the object form of the code would have been unable to obtain the source code. Since the GNU GPL 3.0 permits online distribution of source code relating to object code distributed offline, on the basis that Internet access is now more prevalent, although far from universal, some might argue that the failure to permit online distribution alone under the terms of the GNU GPL 2.0, whilst legally accurate, is anachronistic, and that, as a result, online distribution of source code in respect of object code distributed offline should be considered sufficient. However, whilst there is merit in

6 Where a distributor provides object code via online access, the distributor is not entitled to rely solely on a written offer to supply a copy of source code on a physical medium, per s6, GNU GPL 3.0.

7 per s6(b)(2), GNU GPL 3.0

8 One could make source code available online in addition to making a written offer to supply source code on a physical medium, if one chose to do so.

this argument, since the wording of the licence remains unchanged, a distributor adopting this approach would, on a strict legal interpretation of the licence, be exposing itself to risk[9].

In conclusion, whilst the GNU GPL 3.0 permits mere online distribution of source code, under certain conditions, and the GNU GPL 2.0 permits online distribution of source code where object code is distributed online, online availability of source code is insufficient for offline distribution of object code. However, in reality, it is unclear whether enforcement action would be taken against a distributor of GNU GPL 2.0 covered code, which distributed source code solely from an online location, but was otherwise compliant with the licensing terms.

Using a third party to satisfy source code distribution obligations

On the basis of the preceding section, under some circumstances, a distributor is entitled to place corresponding source code online, for download by interested recipients. However, what of the situation in which the distributor wishes to rely on a third party to make the source code available? For example, the situation in which a distributor seeks to have its supplier publish the corresponding source code, in respect of object code embedded on a device supplied to the distributor.

The intention of each of the GNU GPL 2.0 and 3.0 is to ensure that the source code is available to anyone who receives (whether from the original distributor or otherwise) the object code. In other words, the Freedoms pertaining to the covered code need to be preserved. With this in mind, and with no express preclusion within the licences, the actual distributor of the source code is immaterial, and a distributor of object code can agree with a third party to provide the corresponding source code on its behalf.

In doing so, however, the distributor of the object code does not transfer its liability as against the owner of the copyright in the code, and, potentially, the recipient of the object code; were the third party with whom the distributor has contracted to fail to provide the source code, the distributor would be liable for the breach of its sub-contractor. As such, a distributor would be advised to consider backing off the risk with the third party, on a contractual basis, in addition to conducting appropriate due diligence to ensure, as far as possible, that the third party's hosting will be reliable. The distributor should also perform a test download of the source code being hosted by the third party, to ensure that it is the compliant corresponding source code.

Conclusions

Whilst a distributor of code licensed under the GNU GPL 2.0 and 3.0 may provide an electronic copy of the relevant licence text along with the covered code, in a manner which befits the product in question, mere reference to an online copy of the licence is insufficient, since it fails to show the licence to the recipient.

9 If it wished to mitigate the potential impact of this risk, key sections of the community, particularly those involved in licence compliance and enforcement, could make public statements that they will not take enforcement action against a distributor which made its source code available online, without a written offer to provide source code on a physical medium, to give a degree of comfort. However, the absence of a central licensing body for Free software might restrict the success of such an approach.

A distributor of code under the GNU GPL 3.0 is permitted to make the corresponding source code solely available online whether object code distribution is online or offline, but, if wishing to remain within the wording of the licence, a distributor of code under the GNU GPL 2.0 may only make source code available online where object code distribution is online, or else in addition to providing a written offer or else accompany the object code distribution with source code; mere online distribution of source code does not satisfy the wording of the GNU GPL 2.0 in respect of object code distributed offline.

A distributor may procure a third party to satisfy its obligations under either licence, but the distributor remains liable to the copyright owner, and, potentially, a recipient of the code. As such, a distributor seeking to place reliance on a third party should ensure that it is appropriately protected, in the event that a breach by the third party triggers enforcement action against the distributor.

About the author

Neil Brown *works as a solicitor for Vodafone Group. He is particularly interested in the advancement of knowledge within society, and the role played by copyright in this process. He can also juggle three machetes whilst blindfolded, which is more interesting.*

Copyright in Open Source Software – Understanding the Boundaries

Omar Johnny,[a] *Marc Miller,*[b] *Mark Webbink*[c]

(a) Student, New York Law School; (b) Student, New York Law School; (c) Visiting Professor of Law, New York Law School.

DOI: 10.5033/ifosslr.v2i1.30

Abstract
Copyright ownership tends not to be an issue in closed-source, software development. In that model an individual or business owns — or in-licenses — the copyright in all of the code used in the software application, licenses it to end-users under a binary-only license, and relies on a combination of copyright and trade secret law to enforce contractual rights in the code. By contrast, when software is developed in an open source model, copyright issues abound, and many of these copyright issues are not well understood by software developers. This lack of understanding can undermine the intent of the developers and can potentially lead to unattractive outcomes. As early as the launch of conceptual design in open source software, issues can arise as to ownership of the work and its progeny. When a wide range of hands can touch the open source code, ownership and rights in the code can become blurred. Moreover, not all code contributions to an open source project will be protected by copyright. This paper seeks to explore the application of U.S. copyright law to software, and particularly software that is developed and licensed under an open source model. We address the boundaries of copyright protection and ownership, the importance of intent, timing and creative expression in determining these boundaries, and provide guidance to those looking to launch open source projects.

Keywords
Law; copyright; software

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

Software Development and Copyright

Intent, timing, and creative expression are the three themes that are central to the analysis that will follow. Each is fundamental to the application of U.S. copyright law to the development and ownership of software. Intent refers to the intention of multiple parties as they choose to collaborate and share code. Is it their intent to create a single work or multiple works? Is it their intent to permit reuse of code or place restrictions on such use? Timing addresses the time at which they manifest their intention. Is it before the work commences? Is it after one party has already written some code? Is it after all of the code has been developed? Finally, creative expression goes to the question of whether the code is actually eligible for copyright expression.

Before delving into the issues of intent, timing and creative expression, it will be helpful to understand and appreciate a wide range of development scenarios that can arise in open source development. Every day new open source projects arise, and every day the developer or developers in such projects must make decisions as to who will own the copyright in the code, who will decide how it is to be licensed, what licensing scheme will be used, and, rarely addressed, the limits of those copyright claims and license rights. Following are some of the more typical development scenarios that can be observed in open source software development. In these scenarios, a reference to a sole developer can mean either an individual or a single business entity, for example, a corporation.

Software Development Scenarios

The Sole Developer

This is probably the easiest scenario to understand since it consists of a sole developer who independently works on developing software. Assuming the developer writes all of his/her own code, this individual is the owner of the original work and is still capable of producing or authorizing a derivative work of that original. As we will see, even where the sole developer utilizes pre-existing code the developer will hold rights in that original expression contributed by the sole developer to modify the pre-existing work and to the ultimate product consisting of the combination of pre-existing works and her separate contributions. The rights she elects to share with others in her copyright in this code arise at the sole discretion of this individual developer. Intent and timing play no role as there is only one party involved. Only creative expression remains a factor in determining to what the sole developer has an interest. Where the sole developer is an individual this scenario is most frequently seen at the module level rather than in full blown applications. Examples where the developers were business entities can possibly be seen in the original development work of entities like MySQL[1] and JBoss[2] where employees of the entity did all of the original development work.

1 "MySQL was originally founded and developed in Sweden by two Swedes and a Finn: David Axmark, Allan Larsson and Michael "Monty" Widenius, who had worked together since the 1980's." http://www.mysql.com/about/ Last visited March 29, 2010.

2 "Marc Fleury started the JBoss project in 1999 in order to advance his middleware research interests." http://en.wikipedia.org/wiki/JBoss_%28company%29 Last visited March 29, 2010.

Sole Initial Developer with Others Contributing Code After Initial Development

In this scenario an original developer develops the original code and, as in the first scenario, holds all rights in the code, including the right to determine the licensing of the code. However, after the code is released to the public, the original developer invites others to make contributions to the code. In this instance the intent of the original developer in opening the development process is frequently manifested in the open source license that original developer has used to make the code available. But what if the original developer wants to retain a unified copyright in the entire code base, including the code developed by others? This is the situation faced later by companies such as MySQL and JBoss. It can also be seen in the Netscape/Mozilla development where Netscape developed its browser as a traditional, proprietary software application but later opened the development process to others and permitted them to contribute code – Mozilla.[3]

One or More Developers Agreeing to Develop Jointly in Advance of Development Work

This scenario is most descriptive of the jointly planned open source project. Here the independent developers see a need and come together to develop a strategy around the need. Perhaps they have developed a standard and need to develop a reference implementation of that standard. In any case, prior to the commencement of the work the parties agree that they intend to create a single work to which they will all contribute, and it is their intent that all contributing parties will have a right to exploit the work. It is also possible that they will form a new entity to be the home for the work. In either of these approaches, the intent and timing of the various parties at the time the work commences becomes important to determining the interests in the code, with creative expression remaining a limiting factor in determining rights in the code.

Joint Developers Who Invite Others to Join After Initial Development Work

In this scenario multiple independent developers agree to jointly develop the code as in the scenario above. However, in this instance they invite a new developer to the project after initial development work has been complete. Both the initial developer(s) and the subsequent joining developer intend to create a single, unified work. While the intent is the same as in the previous scenario, the timing of that intent differs. Creative expression remains a limiting factor in determining rights in the code.

One or More Developers Agreeing to Develop Jointly After Initial Development by Each

Multiple independent developers work independently on distinct software modules. After the initial development work is done by each developer, they see the benefit of combining their individual works into a single software application. All parties agree that they intend to create a single, unified work, but that intent is not expressed at the outset of the individual works, only after the works have been created. Thus, timing of the intent becomes a factor along with the

3 January 1998 was also the month that Netscape started the open source Mozilla project. Netscape publicly released the source code of Netscape Communicator 4.0 in the hopes that it would become a popular open source project. It placed this code under the Netscape Public License, which was similar to the GNU General Public License but allowed Netscape to continue to publish proprietary work containing the publicly released code. http://en.wikipedia.org/wiki/Netscape Last visited March 29, 2010.

creative expression in determining rights in the code.

Other Development Activities

While the scenarios above are intended to describe a variety of discrete projects, activities can occur within those development scenarios that further impact issues of intent, timing and creative expression. Among those are:

- Sequential versus simultaneous – or parallel – development
- Borrowing code from other projects
- Partial rewrites – bug fixes
- Complete rewrites incorporating concepts from earlier code

The scenarios and activities described above may not describe every possible permutation of open source software development activity, but they should be sufficient to impart the importance of intent, timing and creative expression in developing open source software. Before turning to why the factors of intent, timing and creative expression are so important in this context we first need to have a clear understanding of ownership interests in copyright.

Ownership of Copyrights in Software

Section 102 of the U.S. Copyright Act – the "Act" – provides copyright protection for "original works of authorship fixed in any tangible medium of expression."[4] Nothing more is required. Thus, whatever ownership interests exist in the copyright commences with the reduction of the expression to some tangible medium, whether paper or an electronic file.

Sole Developer

Section 201(a) of the Act provides that copyright protection in a work "vests initially in the author or authors of the work."[5] In the case of a sole developer, copyright in the software code will vest with her as the sole owner upon fixation of the code in a tangible medium of expression. That is, once she types and saves the code, fixing it in the computer's memory, she is now the proud owner of a brand new copyright in that code that will last the duration of her life, plus an additional 70 years after her death.[6] Nothing more is required of the developer to own the copyright.

Works for Hire

In the case of a developer writing software code within the scope of her employment, the resulting work is known as a "work made for hire." With a work made for hire, "the employer or other person for whom the work was prepared is considered the author for purposes of [the Act]," and

4 17 U.S.C. § 102(a).
5 17 U.S.C. § 201(a).
6 17 U.S.C. § 302(a).

the owner of the copyright in the work.[7] The developer and her employer may agree otherwise, where ownership remains with the developer, through an express agreement in a written instrument signed by both parties.[8] The intention of the developer and her employer, expressed before the code is written, will dictate who owns the copyright in the resulting work.

The result changes where the software developer is not an employee. Software is not a form of copyrightable material that statutorily falls under the list of special order or commissioned works within the scope of works made for hire unless it constitutes a commissioned contribution to a collective work.[9] Thus, the transfer of ownership of software produced outside of the scope of employment must be supported by express contractual language in writing.[10]

Joint Ownership

Section 201(a) of the Act further provides that "[t]he authors of a joint work are co-owners of copyright in the work."[11] Joint authorship in a work arises when "a work is prepared by two or more authors *with the intention that their contribution be merged into inseparable and interdependent parts of a unitary whole [emphasis added]*."[12] The hallmark of joint authorship is the authors' "joint laboring in furtherance of a preconcerted common design."[13] That is, each contributor must intend for their contributions to be merged, however, the contributors are not necessarily required to "work in physical propinquity, or in concert, nor that the respective contributions made by each joint author must be equal in quantity or quality."[14] Furthermore, it is not necessary that the contributors expressly agree, in writing, to create a joint work.[15]

The touchstone of joint authorship is the intention of the joint authors that their contributions be merged at, or before, the moment in time when the contribution of each joint author is created.[16] That is, two developers, who intend that the code they contribute to a project be merged into inseparable and interdependent parts of a unitary whole must express that intent before development commences, and each will own an equal and "undivided ownership in the entire work."[17]

So long as the intent to create a joint work exists prior to the commencement of work, it is even possible for joint authorship to occur "even though the joint authors do not work together in their common design, do not make their respective contributions during the same period, and indeed even if they are complete strangers to each other."[18] This situation is common among developers working independently in developing software modules that are to be included in a unified open-source project. The intent and timing of these developers will dictate who owns the copyright in

7 17 U.S.C. § 201(b).
8 *Id.*
9 17 U.S.C. § 101.
10 17 U.S.C. § 204.
11 17 U.S.C. § 201(a).
12 *See,* 17 U.S.C. § 101.
13 1-6 Melville B. Nimmer, Nimmer on Copyright § 6.03.
14 *Id.*
15 *See, Id.*
16 1-6 Nimmer §§ 6.02, 6.03.
17 *Id.*
18 1-6 Nimmer § 6.03.

the resulting work.

The joint authors are the co-owners of a single copyright in the joint work.[19] However, joint authorship is not the only means by which joint ownership of a work may arise.

> *A joint work will result under any one of the following circumstances: (1) if the work is a product of joint authorship; (2) if the author or copyright proprietor transfers such copyright to more than one person; (3) if the author or copyright proprietor transfers an undivided interest in such copyright to one or more persons, reserving to himself an undivided interest; (4) if upon the death of the author or copyright proprietor, such copyright passes by will or intestacy to more than one person; (5) if the renewal rights under the Copyright Act or the terminated rights under the termination of transfers provisions, vest in a class consisting of more than one person; (6) if the work is subject to state community property laws.*[20]

Each co-owner of a joint work "obtains an undivided ownership in the whole of the joint work, including any portion thereof."[21] In other words, each co-owner may use or license the work, without the consent of other co-owners, in any way she may wish.[22] Co-owners of a copyright do, however, owe to each other a duty to account for any income derived from their use or license of the work.[23]

The Derivative Work

The Act also protects derivative works and compilations.[24] A derivative work is a work based in whole, or in substantial part, upon a pre-existing work, and recasts, transforms, or adapts the underlying work in some way.[25] For copyright protection to extend to a derivative work, "the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution."[26] Since most new works are influenced, in some way, by a pre-existing work, there exists a fine line between a derivative work and an entirely new work.[27] If a developer uses very little of an pre-existing work, taking only code not protected by copyright – like a basic function, or if she uses the pre-existing code in such a way that the resulting program is substantially different from the original, the new creation is

19 17 U.S.C. § 201(a).
20 1-6 Nimmer § 6.01.
21 1-6 Nimmer § 6.06[A].
22 *Id.*
23 *Id.*
24 17 U.S.C. § 103(a).
25 17 U.S.C. § 101
26 1-3 Nimmer § 3.03[A] (citing Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 348, 111 S. Ct. 1282, 113 L. Ed. 2d 358 (1991); Siegel v. Time Warner Inc., 496 F. Supp. 2d 1111, 1151 (C.D. Cal. 2007); Sherry Mfg. Co. v. Towel King of Fla., Inc., 753 F.2d 1565 (11th Cir. 1985); Montgomery v. Noga, 168 F.3d 1282, 1290 n.12 (11th Cir. 1999); Moore Pub., Inc. v. Big Sky Mktg., Inc., 756 F. Supp. 1371, 1374, 1378 (D. Idaho 1990)).
27 Lothar Determan, *Dangerous Liasons – Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs Under Copyright Law, Commercial Licenses, and the GPL*, 21 Berkeley Tech. L. J. 1421, 1430 (2006).

simply a new, original work of authorship and not a derivative work.[28]

Ownership of a separate copyright in a derivative work not only requires more than a minimal contribution to the prior work, but also permission from the owner or owners of the copyright in the prior work. Even with such permission, the creator of the derivative work will own the copyright in only that portion of the derivative work he contributed and not in any portion of the pre-existing work upon which the derivative work is based.[29]

Compilations and Collective Works

Finally, the Act protects interests in compilations, including collective works.[30] A compilation is "a work formed by the collection and assembling of preexisting materials or of data that are selected, coordinated, or arranged in such a way that the resulting work as a whole constitutes an original work of authorship."[31] The preexisting materials or data incorporated in a compilation may or may not, on their own, be capable of being protected by copyright.[32] For example, a program created by stringing together a set of basic functions – which in themselves are not protected by copyright – may receive copyright protection in the selection, coordination, and arrangement of such basic functions.

Those compilations that do, however, incorporate preexisting material or data capable of receiving copyright protection are known as collective works.[33] The Act defines a collective work as "a work, such as a periodical issue, anthology, or encyclopedia, in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole."[34] For example, a program that includes preexisting modules – which in themselves protected by copyright – may receive copyright protection in the selection, coordination, and arrangement of the modules.

As with derivative works, for copyright protection to extend to a compilation or collective work, "the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution."[35] In the case of a collective work, the copyright in the prior work and copyright in the collective work as a whole are separate and distinct; the author of the prior work retains copyright ownership in her work, while ownership of the collective work, including contributions made by the author of the collective work, vests in the author of the collective work.[36]

Joint Works vs. Derivative Works vs. Compilations

28 *Id.*
29 1-6 NIMMER, § 6.05.
30 17 U.S.C. § 103(a).
31 *Id.*
32 1-3 NIMMER § 3.02.
33 *Id.*
34 17 U.S.C. § 101.
35 1-3 NIMMER § 3.03[A] (citing Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 348, 111 S. Ct. 1282, 113 L. Ed. 2d 358 (1991); Siegel v. Time Warner Inc., 496 F. Supp. 2d 1111, 1151 (C.D. Cal. 2007); Sherry Mfg. Co. v. Towel King of Fla., Inc., 753 F.2d 1565 (11th Cir. 1985); Montgomery v. Noga, 168 F.3d 1282, 1290 n.12 (11th Cir. 1999); Moore Pub., Inc. v. Big Sky Mktg., Inc., 756 F. Supp. 1371, 1374, 1378 (D. Idaho 1990)).
36 17 U.S.C. § 201(c).

Joint works of authorship share similarities with derivative works and compilations and, but for the intention of the authors, could be seen as one and the same.[37] Depending on the intent at the time of creation, one author's recasting, transforming, or adapting of another author's preexisting work may create either a derivative work or a joint work consisting of inseparable parts.[38] Similarly, depending on the intent at the time of creation the assembling of the works of several different authors into a collective whole may create either a compilation or a joint work consisting of interdependent parts.[39] Which case applies in each instance "lies in the intent of each contributing author at the time his contribution is written."[40] If, at the time of creation, the author intends his contribution and those contributions of others "be merged into inseparable or interdependent parts of a unitary whole" then such a merger creates a joint work.[41] On the other hand, if the intention to merge occurs only after creation of the work, then such a merger results in a derivative work or a compilation.[42]

Software Development Scenarios – The Ownership Issues

The Sole Developer

In the case of a sole developer, copyright ownership in the her work will vest with her as the sole owner. Creative expression here is vital; if the sole developer works independently, writing all of her own code, the resulting program is an original work of authorship all her own. However, if the sole developer bases her program on pre-existing code, as most software developers will inevitably do, then she toes a fine line between original works of authorship and derivative works. If the sole developer uses the pre-existing code protected by copyright[43] in such a way that the program she develops is substantially different from the original program, she will own the copyright in the original work of authorship. However, copyright ownership may vest in the sole developer as the owner of a derivative work if her work is based on a prior protected work in whole, or in substantial part, and she recasts, transforms, or adapts the underlying work in some way that exhibits more than a minimal contribution. It is important to remember that in the case of a derivative work the sole developer must have received permission from the upstream owner of the prior protected work and that ownership of a copyright in the resulting derivative work has no effect upon the copyright ownership of the prior protected work.

Similarly, copyright ownership may vest with the sole developer using a software development kit, with pre-packaged modules, to create her program. If she merely selects, coordinates, organizes, and arranges the pre-existing modules in a particular fashion, the resulting work will receive "thin" copyright protection as a compilation; conversely, if she adapts, transforms, or recasts the pre-existing modules in some way that exhibits more than a minimal contribution, the resulting work will receive copyright protection as a derivative work.

37 1-6 NIMMER § 6.05.
38 *Id.*
39 *Id.*
40 *Id.*
41 *Id.*
42 *Id.*
43 We refer here to works protected by copyright or "protected works" to emphasise the fact that not all software code will be protected by copyright. Our discussion of code not protected by copyright follows later in the paper.

Sole Initial Developer with Others Contributing Code After Initial Development

In this scenario, where the sole developer releases her program to the public and invites others to make contributions to the code, the sole developer retains ownership in the original code. The sole developer has the exclusive right to license the use of her work to other contributors. "Under the conventional doctrine of derivative and collective works," the sole developer retains her copyright in the original code, while the downstream contributors may claim exclusive ownership over the copyright in the derivative work, subject to the license to use the underlying work.[44] In this case, the sole developer, "does not obtain any property right in the derivative (or collective) work, and likewise the downstream contributor does not obtain any property right in the underlying work."[45]

The sole developer may modify the conventional doctrine of derivative and collective works and retain a unified copyright in the entire code base, including the code developed by others, by requiring modifications and enhancements to be assigned back to her as the owner of the copyright in the main code base. This is the approach taken by the GNU projects run by the Free Software Foundation.[46] This will ensure that the entire open source project may be protected as a whole by a single copyright owner, instead of having several copyrights in different modules, owned by different developers.[47]

One or More Developers Agreeing to Develop Jointly in Advance of Development Work

Where one or more developers agree to jointly develop an open source project, the copyright issues may be fairly straightforward. When independent developers see a need and come together to develop a strategy around that that need, their actions give rise to a joint work of authorship. This scenario falls squarely within the Act's definition of a joint work – a work "prepared by two or more authors with the intention that their contributions be merged into inseparable or interdependent parts of a unitary whole."[48] Independent developers jointly labouring in furtherance of a pre-concerted common design is the hallmark of joint authorship.[49] The developers are not required to work in physical proximity to each other – in fact, they may be complete strangers to one another – nor are the developers required to contribute equal portions in quantity or quality, nor are the developers required to develop their contributions simultaneously. All that is required for joint authorship is that the developers intend that their contribution be merged before, or at the moment in time when the code is developed.

Furthermore, the developers need not memorialise their intention to merge their contributions in an

44 1-6 NIMMER § 6.06[B]. Remember, if the downstream contributor is an unauthorised user of the underlying code (meaning the terms of the license were broken), that downstream contributor receives no copyright protection for the resulting work and is liable to the sole developer for copyright infringement.

45 *Id.*

46 Eben Moglen, *Why FSF Gets Copyright Assignment From Contributors*," http://www.gnu.org/licenses/why-assign.html Last visited March 30, 2010.

47 Sun Microsystems requires, through the open source license, that developers contributing to the OpenOffice.org project assign their copyright back to Sun and that all contributions to the source code are required to be made automatically available under the same open source license. *See*, OPENOFFICE.ORG, *FAQs – Licensing*, http://www.openoffice.org/FAQs/faq-licensing.html#sca1 Last visited March 30, 2010.

48 17 U.S.C. § 101.

49 1-6 NIMMER § 6.03.

express agreement.[50] However, it would be wise of the developers to draw up an agreement, perhaps in the form of a license, and clearly lay out the terms of the development plan for all to see. Remember that each developer, as a joint owner, has an undivided interest in the entire project and has the right use or exploit the work without consent of the other joint owners, subject only to the duty to account.[51] When memorialising their agreement in writing, the joint developers may chose to form a new entity for the purposes of holding the unified copyright, or they may impose certain restrictions and limitations on the use of the work by future downstream contributors through one of the several open source licenses.

Joint Developers Who Invite Others to Join After Initial Development Work

This scenario is much like the previous scenario, where independent developers agree to jointly develop an open source project. However, the sole differentiating fact is that the initial group of developers subsequently invites new developers to work on the project after the initial development work has been completed. Regardless of that factual distinction, the copyright ramifications are no different – so long as both the initial developer(s) and the subsequent joining developers intend "that their contributions be merged into inseparable or interdependent parts of a unitary whole," they will be considered co-owners of a joint work of authorship.[52]

Despite the fact that the timing of the developers' manifestation of their intent differs, their intent to have their contributions merged into a single unitary whole carries the day. Courts have held, "that the design of collaboration between joint authors need be pre-concerted only in the sense that at the time each author makes his contribution he intends that it shall be an integrated part of a greater work with supplementary contributions to be made by one or more other authors."[53] In other words, the initial developers' intent to jointly develop an open source project that would include future contributions by other developers is sufficient to create a joint work of authorship so long as the subsequent developers also manifest their intent to contribute to a unified. work. "The fact that the identity of such other authors has not been determined at the time of the original creation does not, according to these cases, derogate from their status as joint authors.'[54] Because of the potential for ambiguity in a subsequent contribution constituting a part of a joint work or a derivative work, capturing the express intent of the subsequent author at the time of authorship and contribution can be important.

One or More Developers Agreeing to Develop Jointly After Initial Development by Each

In this scenario, one or more independent developers work independently on distinct software modules. Each software module is an original work of authorship and copyright ownership is vested with each individual developer as the sole owner.[55] After the initial development by each developer, they collectively see the benefit of combining their individual works into a single software application. Because the developers' intent and the timing of that intent are different than

50 *Id.*
51 *Id.*
52 *Id.*
53 *Id.* (citing Words & Data, Inc. v. GTE Communications Servs., Inc., 765 F. Supp. 570, 575 (W.D. Mo. 1991).
54 *Id.*
55 17 U.S.C. § 201(a).

in the previous two scenarios, the copyright ramifications, as well, are distinctly different.

Had the independent developers initially, or primarily, intended for their contributions to be "merged into inseparable or interdependent parts of a unitary whole," they would be joint authors of the resulting software application, just as in the above scenarios. However, it is important to remember that each developer solely owns the copyright in her module as an original work of authorship. They are neither necessarily inseparable nor independent. And because the developers manifested their intent to merge the individual modules into a single application well after the modules were developed, the resulting software application is a considered a collective work for copyright purposes.[56] The intent of each contributing developer at the time her contribution is written distinguishes a collective work from a joint work based upon interdependent parts.[57] "If [her] work is written 'with the intention that [her] contribution ... be merged into inseparable or interdependent parts of a unitary whole' then the merger of [her] contribution with that of others creates a joint work. If such intention occurs only after the work has been written, then the merger results in a … collective work."[58]

While the individual developers retain sole ownership in the copyright of their respective individual modules, as simultaneous contributors to the collective work with the intent of creating a single unified collection work, they may each own an undivided interest in the copyright on the software application as a collective work.

Other Development Activities

The above-mentioned scenarios are intended to illustrate the issues over copyright ownership that arise in typical development scenarios. Activities can occur within those development scenarios may further impact issues of intent, timing, and creative expression. Some examples of these activities are:

- Sequential versus Simultaneous – or parallel – development.
- A developer borrowing code from other projects to incorporate into her project, assuming the borrowed code is more than a *de minimis* amount of copyright protected code, would place that developer in the position of creating a derivative work. To receive copyright protection for her work in that case, the developer would need permission to use the underlying work and would need to recast, transform, or adapt the borrowed code in some way.
- A developer working on a partial rewrite of existing code, such as a bug fix, is not likely to receive copyright protection for her contribution. If the modifications are slight in nature, the developer's contributions will fail to meet the minimum threshold of creative expression for copyright protection as an original work of authorship. Similarly, the developer's contributions will fail to meet the threshold for a derivative work since the rewrite does not recast, transform, or adapt the underlying work in a way that constitutes more than a minimal contribution or trivial

56 1-6 NIMMER § 6.05.
57 *Id.*
58 *Id.*

variation.

- Conversely, a developer working on a complete rewrite that incorporates concepts from the original work will likely receive copyright protection as either an original work of authorship or a derivative work. If the developer uses the original work in such a way that the rewrite she develops is substantially different from the original program, she may own the copyright in the original work of authorship. However, copyright ownership of the rewrite may vest in the developer as the owner of a derivative work if her rewrite is based on a prior work in whole, or in substantial part, and she recasts, transforms, or adapts the underlying work in some way that exhibits more than a minimal contribution.

Software Code Not Subject To Copyright

Open source software developers may assume every line of code they write is protected by copyright such that when they apply an open source license to that code, they are under the impression that the license will govern all use of the code. However, under U.S. copyright law, not every line of code receives copyright protection. It is important to understand why that is the case, and how it impacts enforcement rights. Assuming *arguendo* that a violation of an open source license constitutes copyright infringement, one need understand how U.S. courts will approach the issue of copyright infringement in software code.

Abstraction, Filtration, Comparison Test

When presented with copyright infringement in software code, different U.S. Federal Circuit Courts apply different tests. The most broadly adopted of these tests is the abstraction, filtration and comparison test – the “AFC test” – as first adopted by the Second Circuit. A few Circuits have adopted narrower versions of the AFC test, a handful have expressly rejected the AFC test in favor of a still narrower standard, and some have yet to adopt any definition of derivative works in software.[59]

Given the dominance of the AFC test we will focus our attention to its application. AFC is a test for substantial similarity of a computer program structure. Under this approach a court first breaks down the allegedly infringed program into its structural parts.[60] This process of breaking down the alleged infringing program into various levels or layers is the abstraction part of the test. It is a means of separating the program by its various levels – *abstractions* – to determine where there could be possible infringement. The abstractions test progresses in order of “increasing generality” from object code, to source code, to parameter lists, to services required, to general outline.[61]

“[Next, the court examines] each of these parts for such things as incorporated ideas, expression that is necessarily incidental to those ideas, and elements that are taken from the public domain;”[62]

59 Dan Ravicher, *Software Derivative Work: A Circuit Dependent Determination,* 1, (Nov. 2002).

60 *See, Computer Assoc.*, 982 F.2d at 706.

61 *See*, *Computer Associates Intern., Inc. v. Altai, Inc.*, 775 F. Supp. 544, 560 (E.D.N.Y. 1991).

62 *Id.*

in other words, the court is seeking to identify those components within a particular level that are not protected by copyright. "[The] court would then be able to sift out all non-protectable material."[63] This process is the filtration part of the test. It is a process for identifying and removing code that is not protected under the Act from that material which is subject to copyright protection. Examples of elements or code not subject to copyright protection are functional elements, merger of expression and idea, scènes à faire in software, facts in software, and code in the public domain.

"Left with a kernel, or possibly kernels, of creative expression after following this process of elimination, the court's last step is to compare the protectable material with the corresponding sections of an allegedly infringing program."[64] This is known as the comparison part of the AFC test. "The result of this comparison will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of infringement."[65]

Consider the following example of the AFC test for infringement. The holders of the copyright in Busybox claim the program Tools[66] has copied some portion of the Busybox code and is in violation of the license governing Busybox - the GNU General Public License version 2. The first level of abstraction, or layer of review, would be the entire source code for Busybox.[67] A secondary layer would be the various files that are contained at the next level of that Busybox source, e.g., applets, arch, archival AUTHORS, Config.in, e2fsprogs, README, etc. A tertiary layer would be the actual source files, like executables, header files, or text files that contain program instructions or information. The final layer would be the actual lines of code or text. Where the court starts its analysis will largely be determined by the extent of alleged infringement, and each layer of abstraction is reviewed for both literal and non-literal copying

In our hypothetical the court determines that the sole basis for the infringement claim lies within a file in Busybox named e2fsprogs. Saying that there is an infringement of this file within Busybox does not necessarily mean that the alleged infringing party infringed the entire file. If the entire file is alleged to infringe, the court will assess the infringement at two levels, the structure and sequence of the entire e2fsprogs file, and then any sublayers. To consider the sublayers the court abstracts the e2fsprogs file into its various sub layers. In e2fsprogs the court will consider each of the component files, such as the e2fs_lib.h file, a header file. At this level there are no longer any subfiles to abstract. Within e2s_lib.h – at the file level – the court would examine whether the actual lines of code or text within the header file constitute material protected by copyright. This is the filtration test. We next consider the various considerations that may be applied by a court in determining protectable versus non-protectable material.

Literal vs. Nonliteral Copying

With respect to such things as musical, dramatic, and motion picture works, and works of "literature," as contrasted with "literary" works in the broader statutory sense, to the extent that

63 *Id.*
64 *Id.*
65 *Id.*
66 "Tools" is a fictitious program and should not be construed to be any real program.
67 The source for Busybox may be found at http://git.busybox.net/busybox/snapshot/busybox-1_15_3.tar.bz2 Last viewed on March 1, 2010.

such a work contains original, literal manifestations, the work is protected by copyright."[68] However, a work in one form for may infringe the same work expressed in a different form even if it does copy word for word. For example, a motion picture may infringe a book by using "the story's unique setting, characters, plot, and sequence of events."[69] This is nonliteral copying. "This type of copying of nonliteral expression, if sufficiently extensive, has never been upheld as permissible copying; rather, it has always been viewed as copying of expressive elements of creative originality."[70]

In *Lotus* the court recognized the amorphous nature of "nonliteral" elements of computer programs. [71] "Unlike the written code of a program or a flowchart that can be printed on paper, nonliteral elements – including such elements as the overall organization of a program, the structure of a program's command system, and the presentation of information on the screen – may be less tangibly represented."[72] "In the context of computer programs, nonliteral elements have often been referred to as the "look and feel" of a program."[73]

The *Lotus* court's conclusion is consistent with the treatment of the user interface and some other nonliteral aspects of computer programs, which are not merely articles "having an intrinsic utilitarian function."[74] When computer programs include both literal and nonliteral elements, which can be identified separately from and are capable of existing independently of the utilitarian aspects of the program, they are potentially copyrightable.[75]

Because the court must determine the scope of copyright protection that extends to a computer program's nonliteral structure,[76] the *Computer Associates* court held that comparison, the third and final step of the abstraction, filtration, comparison test for substantial similarity, is appropriate for nonliteral program components.[77]

Nimmer warns of the pitfalls in use of a "look and feel" type of test.

> *More broadly, the touchstone of "total concept and feel" threatens to subvert the very essence of copyright, namely the protection of original expression. "Concepts" are statutorily ineligible for copyright protection; for courts to advert to a work's "total concept" as the essence of its protectible character seems ill-advised in the extreme. Further, the addition of "feel" to the judicial inquiry, being a wholly amorphous referent, merely invites an abdication of analysis. In addition, "total concept and feel" should not be viewed as a sine qua non for infringement--similarity that is otherwise actionable cannot be rendered defensible simply because of a different*

68 *See*, *Lotus Dev. Corp. v. Paperback Software Int'l*, 740 F. Supp. 37, 51 (D. Mass. 1990).
69 *Id.* at 52, quoting Stewart v. Abend, 495 U.S. 207, 109 L. Ed. 2d 184, 110 S. Ct. 1750, 1759, 14 U.S.P.Q.2D (BNA) 1614 (1990).
70 *Id.* at 52.
71 *Id.* at 46.
72 *Id.*
73 *Id* at 62.
74 17 U.S.C. § 101 (defining "useful article").
75 *Lotus*, 740 F. Supp. at 54.
76 *See*, *Computer Assocs. Int'l v. Altai*, 982 F.2d 693, 703 (2d Cir. N.Y. 1992).
77 See, *Id* at 710.

> *"concept and feel." In sum, therefore, the frequent invocations of this standard do little to bring order to the inquiry into what constitutes substantial similarity, and would be better abandoned.*[78]

However, the Ninth Circuit ultimately defended the standard against Nimmer's critique:

> *Some commentators have worried that the "total concept and feel" standard may "invite an abdication of analysis," because "feel" can seem a "wholly amorphous referent." . . . But the [Ninth Circuit's] caselaw is not so incautious. Where [the court] has described possible infringement in terms of whether two designs have or do not have a substantially similar "total concept and feel," the court generally has taken care to identify precisely the particular aesthetic decisions--original to the plaintiff and copied by the defendant--that might be thought to make the designs similar in the aggregate.*[79]

Functionality Exception to Copyright Protection

When developing computer programs it is inevitable that some of the code will be functional in nature. As stated earlier, the Act awards copyright protection to creative expression. "Functional elements and elements taken from the public domain do not qualify for copyright protection."[80] Therefore, there is no striking similarity even between two identical works so as to warrant an inference of copying to the extent that, albeit copyrightable, functional considerations can account for the identity.[81]

What makes an element "functional?" Elements are functional if they are necessary to the program and do not exhibit any creativity. Aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact are functional in nature and, thus, not protected by copyright.[82]

Functional elements may also be dictated by the nature of the program being developed. In *Computer Associates*, "the district court found that the overlap exhibited between the list of services required for both ADAPTER and OSCAR 3.5 was determined by the demands of the operating system and of the applications program to which it was to be linked through ADAPTER or OSCAR."[83] These aspects of the program's structure are therefore functional in nature and not copyrightable.

For example, graphical user interfaces [GUI's] generated by computer programs are partly artistic and partly functional. They are a tool to facilitate communication between the user and the computer. GUIs do graphically what a character-based interface, which requires a user to type in

78 4-13 Nimmer § 13.03[A][1].

79 *Tufenkian Import/Export Ventures, Inc. v. Einstein Moomjy, Inc.*, 338 F.3d 127, 134 (2nd Cir. 2003).

80 *Computer Assoc.*, 982 F.2d at 714.

81 *See,* 4-13 Nimmer §13.02[B].

82 *See, Computer Assoc*., 982 F.2d at 715.

83 *Id.*

alphanumeric commands, does manually.[84]

In *Lotus* the court held that the Lotus menu command hierarchy is an uncopyrightable method of operation. [85]

> *The Lotus menu command hierarchy provides the means by which users control and operate Lotus 1-2-3. If users wish to copy material, for example, they use the Copy command. If users wish to print material, they use the Print command. Users must use the command terms to tell the computer what to do. Without the menu command hierarchy, users would not be able to access and control, or indeed make use of, Lotus 1-2-3's functional capabilities.*[86]

The menu command hierarchy in Lotus 1-2-3 is functional by nature of the program and therefore not copyrightable.[87]

Other areas to consider when determining whether an element is purely or primarily functional include:

- hardware standards;
- software standards;
- computer manufacturer design standards;
- target industry practices; and
- computer industry programming practices.[88]

Idea/Expression Merger Exception to Copyright Protection

Under the Act, in no case does copyright protection extend to any idea regardless of the form in which it is described, explained, illustrated, or embodied in such work.[89] "It is a fundamental precept of copyright that only the expression of ideas, and not the ideas themselves, are copyrightable."[90] "Merely stating the rule, however, does not make any easier the task of drawing the line between where idea ends and expression begins."[91]

> *The line between idea and expression may be drawn with reference to the end sought to be achieved by the work in question. In other words, the purpose or function of a utilitarian work would be the work's idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea…Where there are various means of achieving the*

84 *Apple Computer, Inc v. Microsoft Corp.,* 35 F.3d 1435, 1445 (9th Cir. 1994).
85 *See, Lotus Dev. Corp. v. Borland Int'l*,, 49 F.3d 807, 819 (1st Cir. 1995).
86 *Id.* at 815.
87 *See, Id.* at 815.
88 4-13 Nimmer §13.03[F].
89 17 U.S.C. 102(b).
90 1-2 Nimmer §2.02.
91 4-13 Nimmer §13.03[B][2][a].

> *desired purpose, then the particular means chosen is not necessary to the purpose; hence, there is expression, not idea.*[92]

The characteristics of computer software, a utilitarian work, make the determination of idea and expression more complicated. Competitive forces that exist in the software marketplace lead to the problem that multiple programmers may design identical or highly similar works.[93]

> *Efficiency is an industry-wide goal. Since, as we have already noted, there may be only a limited number of efficient implementations for any given program task, it is quite possible that multiple programmers, working independently, will design the identical method employed in the allegedly infringed work. Of course, if this is the case, there is no copyright infringement.*[94]

The merger doctrine is as an exception to the idea-expression dichotomy which holds that, when there are so few ways of expressing an idea, not even the expression is protected by copyright.[95] When idea and expression merge such that a given idea is inseparably tied to a particular expression, rigorously protecting the expression would confer a monopoly over the idea itself, in contravention of the statutory command. To prevent such an occurrence, courts have invoked the merger doctrine.[96]

In the realm of computer programs, merger issues may arise in unusual ways. Although, there may be many ways to implement a particular idea, efficiency concerns can make one or two choices so compelling, as to virtually eliminate any other form of expression.[97]

> *Computer searching and sorting algorithms provide good examples of this phenomenon. Any computer system that deals with significant quantities of data will spend much of its operating time engaged in sorting and searching through that data. Because the amount of time spent on sorting and searching operations can significantly influence a program's operating speed, efficient methods of sorting are highly desirable. A great deal of computer science research has been devoted to developing methods of sorting or searching through data, and to analyzing the relative efficiency of various methods. As a result of such research, it is now recognized that some methods of sorting or searching are significantly more efficient than others in handling particular types of data, even though any of numerous methods will work. In such cases, the merger doctrine should be applied to deny protection to those elements of a program dictated purely by efficiency concerns.*[98]

92 *Whelan Assoc., Inc. v. Jaslow Dental Lab, Inc.*, 797 F.2d 1222, 1236 (3rd Cir. 1986).
93 *Computer Assoc.,* 982 F.2d at 708.
94 *Id.*
95 *See, BUC Int'l Corp. v. Int'l Yacht Council Ltd.*, 489 F.3d 1129, 1143 (11th Cir. 2007).
96 See, 4-13 NIMMER § 13.03[B][3].
97 See, 4-13 NIMMER § 13.03[F][2].
98 *Id.*

While the merger doctrine and the functionality exception to copyright protection are similar, there is a slight difference which distinguishes the two. "Under the merger doctrine, when an idea can be expressed in only one fashion, that expression is not protected by copyright."[99] Here the focus is on the limitations of the expression of an idea which results in the merger of that idea and its expression. In contrast, elements are functional if they are necessary to the program and do not exhibit any creativity.[100] In reference to the functionality exception, the focus is not on the limitations on expression of an idea resulting in merger of the two, but on aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact.[101]

Scènes à Faire in Software Exception to Copyright Protection

The Act does not directly define the scènes à faire doctrine. Scènes à faire refers to aspects of a work that are indispensable or standard parts of the material to be copyrighted.[102] "The [scènes à faire] doctrine is often invoked to immunize from liability similarity of incidents or plot that necessarily follows from a common theme or setting."[103] "Judge Leon Yankwich has called such incidents scènes à faire, *i.e.*, scenes which must be done."[104]

> *As was remarked above concerning merger, this doctrine does not limit the subject matter of copyright; instead, it defines the contours of infringing conduct. Labeling certain stock elements as "scènes à faire" does not imply that they are uncopyrightable; it merely states that similarities between plaintiff's and defendant's works that are limited to hackneyed elements cannot furnish the basis for finding substantial similarity.*[105]

In *Durang,* the court found that alleged similarities that follow obviously from the unprotected idea are therefore unprotected scènes à faire.[106] The *Durang* court held that the lower court properly applied the scènes à faire doctrine to hold unprotectable, forms of expression that were either stock scenes or scenes that flowed necessarily from common unprotectable ideas.[107] The *Durang* court went on to explain that common in that context means common to the works at issue, not necessarily referring commonly found in other artistic works.[108]

Further, under the doctrine of scènes à faire, elements of an original work are not protected if the "common idea is only capable of expression in more or less stereotyped form."[109] "Beyond mere plot incidents applicable to works of fiction, the scènes à faire doctrine can be invoked throughout other copyright contexts as well; from guidebooks to infomercials to Frequently Asked Questions

99 4-13 Nimmer § 13.03[F][2]

100 *See, Computer Assoc.*, 982 F.2d at 715.

101 *See, Computer Assoc.*, 982 F.2d at 715.

102 See, *Id.* at 710.

103 4-13 Nimmer §13.03[B][4].

104 *Id.*

105 *Id.*

106 *See, John William See v. Christopher Durang and LA. Stage Co.*, 711 F.2d 141, 143 (9th Cir. 1983).

107 *Id.*

108 *Id.*

109 *Mist-On Sys. v. Gilley's European Tan Spa*, 303 F. Supp. 2d 974, 978 (W.D. Wis. 2002).

web pages and beyond."[110]

In *Gilley's European Tan Spa*, "[the] plaintiff contended that defendants infringed plaintiff's exclusive rights under the Copyright Act by preparing and displaying on their web page an unauthorized Frequently Asked Questions page that mirrors the Frequently Asked Questions page found on plaintiff's web page."[111]

> *The Gilley's court held a business cannot copyright a Frequently Asked Questions page as such or copyright words or phrases commonly used to assemble any given Frequently Asked Questions page. The format of a Frequently Asked Questions page is a common idea in our society; the elements of a Frequently Asked Questions page (a list of questions beginning with common words) are stereotypical. Some additional similarity beyond generic formatting is necessary to establish infringement."*[112]

Applied to computer programs, the merger and scènes à faire doctrines suggest that if a limited number of options exist to achieve a given function efficiently, interoperate with another application, or run in a given environment, copyright will not permit exclusive control over those program elements.[113] Scènes à faire is distinguishable from the merger doctrine because, the merger doctrine holds that when there are so few ways of expressing an idea, not even the expression is protected by copyright.[114] The idea and expression are in essence, fused. In contrast, scènes à faire relates to alleged similarities that follow obviously from the unprotected idea.[115] The focus in scènes à faire is not on the merged idea and expression or the limited number of ways to express the idea, but on the similarities between expression in question which are a natural result of the idea being expressed.

Moreover, scènes à faire is also distinguishable from the functionality exception to copyright protection. While scènes à faire is expression that relates to stock scenes or elements which are necessary to the idea such as frequently asked questions or "readme" files, functionality relates to aspects of a program's structure which are dictated by the nature of other programs with which they were designed to interact,[116] such as hardware or software standards. As software development languages become more and more sophisticated in the ready-made tools they provide developers and as more and more developers, especially open source developers, reuse standard or stock bits of code to carry out standard functions, we will see the scènes à faire doctrine applied with greater regularity in software to deny copyright protection.

Public Domain Exception to Copyright Protection

Works eligible for copyright protection may nonetheless enter the public domain, i.e., no longer enjoy that copyright protection. For example, a work whose copyright term has expired is

110 4-13 NIMMER § 13.03[B][4].
111 *Mist-On Sys.*, 303 F. Supp. 2d at 976 (W.D. Wis. 2002).
112 *Id.* at 978.
113 *Computer Assocs.*, 982 F.2d at 709-10.
114 *BUC Int'l Corp. v. Int'l Yacht Council Ltd.*, 489 F.3d 1129, 1143 (11th Cir. 2007).
115 *See v. Durang*, 711 F.2d at 143.
116 *Computer Assoc.*, 982 F.2d at 715.

obviously not protected. Similarly, a work may have entered the public domain by reason of the failure to satisfy certain statutory formalities of the Act as it existed prior to 1978. In addition, an author may choose to lift the protections of copyright and voluntary place the work into the public domain.[117] "Moreover, copyright protection under the Act is not available for any work of the United States Government, but the United States Government is not precluded from receiving and holding copyrights transferred to it by assignment, bequest, or otherwise."[118]

What is the public domain? "A work of authorship is in the public domain if it is no longer under copyright protection, it failed to meet the requirements for copyright protection, or the holder of the copyright disclaimed copyright in the work."[119] Works in the public domain are free for anyone to use without permission from the former owners(s) of the copyright.[120] Material found in the public domain is free for the taking and cannot be appropriated by a single author even though it is included in a copyrighted work.[121]

> *An enormous amount of public domain software exists in the computer industry, perhaps to a much greater extent than is true of other fields. Nationwide computer "bulletin boards" permit users to share and distribute programs. In addition, computer programming texts may contain examples of actual code that programmers are encouraged to copy. Programmers often will build existing public domain software into their works. The courts thus must be careful to limit protection only to those elements of the program that represent the author's original work.*[122]

Copyright protection is automatic and vested in the author the moment it is created and fixed in a tangible form.[123] Voluntarily placing a copyrighted work in the public domain requires some manifest expression of the author's intent.[124] Consequently, open source developers should be cautious about assuming code to be in the public domain without some express statement from the copyright holder declaring the code to be in the public domain. An invitation to use with nothing more may be sufficient, but combined with a requirement of attribution suggests the author is merely granting permission to use while retaining the copyright and its various protections. A more definite state, such as "as the author of this work I disclaim the copyright work and declare the work to be in the public domain" would leave little doubt as to the copyright holder's intent. The Creative Commons Copyright-Only Dedication statement gives some indication of the complexity of committing a work to the public domain.[125]

Facts in Software Exception to Copyright Protection

> *Facts, whether alone or as part of a compilation, are not original and*

117 *See*, 1-2 Nimmer § 2.03[G].
118 17 U.S.C § 105
119 http://www.copyright.gov/help/faq/faq-definitions.html Last visited March 30, 2010.
120 *See, Id.*
121 *See, Computer Assocs.*, 982 F.2d at 710.
122 4-13 Nimmer § 13.03[F][4].
123 http://www.copyright.gov/help/faq/faq-general.html, last visited April 3, 2010.
124 4-13 Nimmer § 13.03[F][4].
125 http://creativecommons.org/licenses/publicdomain/ Last visited April 3, 2010.

> *therefore may not be copyrighted. A factual compilation is eligible for copyright if it features an original selection or arrangement of facts, but the copyright is limited to the particular selection or arrangement. In no event may copyright extend to the facts themselves.*[126]

"In no case does copyright protection for an original work of authorship extend to any … discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."[127] Nimmer explains that the discoverer merely finds and records.

> *He may not claim that the facts are original with him, although there may be originality and hence, authorship in the manner of reporting, i.e., the expression, of the facts. As copyright may only be conferred upon authors, it follows that quite apart from their status as ideas, discoveries as facts per se may not be the subject of copyright.*[128]

> *[The Court in Fiest*[129] *noted] the tension between two well-established copyright propositions, … facts are not copyrightable, whereas compilations of facts generally are. As the tool for untangling those disparate strands, the Court relied on the bedrock principle of copyright subsistence--that only original works of authorship qualify for protection. Given that facts, by themselves, are never copyrightable, the Court reasoned that the element of originality that renders a factual compilation protectible must lie in selection, coordination, or arrangement of facts, with the scope of protection concomitantly limited to that original selection, coordination, or arrangement. That formulation, it should be noted, corresponds to the scope of copyright generally for derivative or collective works.*[130]

How does this relate to computer software? In *WIREdata* an owner of a copyright attempted to hide data in its copyrighted program. [131] Specifically, the copyright owner attempted to use copyright law to "block access to data that not only are neither copyrightable nor copyrighted, but were not created or obtained by the copyright owner."[132]

> *The information at issue in [WIREdata] was collected and then was slotted into plaintiff's database. Defendant did not want that database's organized structure; it only wanted the raw data. That last consideration proved decisive in defeating plaintiff's copyright infringement claim: A work that merely copies uncopyrighted material such as facts is wholly unoriginal and the making of such a work is therefore not an infringement of copyright.*[133]

126 *Feist Publ'ns, Inc. v. Rural Tel. Serv. Co*., 499 U.S. 340, 350 - 351 (1991).
127 17 U.S.C. §102(b).
128 1-2 NIMMER § 2.03[E].
129 *Feist*, 499 U.S. at 350 (1991).
130 1-3 NIMMER § 3.04[2][a].
131 *Assessment Techs. of WI, LLC v. WIREdata, Inc*., 350 F.3d 640 (7th Cir. 2003).
132 1-3 NIMMER § 3.04[B][3][a].
133 *Id.*

Within the framework of computer software development it will not be unusual to find lines of code that merely make a factual statement. A reference in a line of code to another place in the program, a table showing equivalences, a target name may all be merely factual statements within the context of the software and, thus, not eligible for copyright protection.

Avoiding Infringement

Fair use

As we have seen, the owner of copyright in software code has the exclusive right to reproduce or to authorize others to reproduce her work. However, that right is subject to certain limitations, one of which is the doctrine of fair use.[134] Originally developed by the courts through case-law, certain uses or reproductions of a work protected by copyright are considered to be fair, and thus, not an infringement of the owner's exclusive rights granted by copyright law. In other words, fair use is a defence to copyright infringement.

Section 107 of the Act contains a list of the various purposes for which the reproduction of a particular work may be considered fair, such as criticism, comment, news reporting, teaching, scholarship, and research.[135] In addition, the Act sets out four factors to be considered by a court determining whether or not a particular use is fair:

1. the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;
2. the nature of the copyrighted work;
3. the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
4. the effect of the use upon the potential market for or value of the copyrighted work.[136]

From a practical perspective, it is important to recognize that the fair use doctrine is malleable – the court has wide discretion in its application of the four factors to the particular facts of the case before it. There are no hard and fast rules in fair use and the difference between an infringing use and a fair use may be murky and not easy to delineate. Using a work protected by copyright without permission poses a substantial amount of risk. But for fair use, the unauthorized use of a work protected by copyright is an infringement. Unless the use falls within one of permissible statutory uses, there is no way to conclusively know whether the use is fair without costly and expensive litigation.

Should a developer choose to roll the dice and rely on fair use as a defence to copyright infringement, she can only stand to benefit from paying close attention to the language of the statute. A safe play is to use the copyrighted work for one of the permissible purposes expressed

134 17 U.S.C. § 107.
135 *Id.*
136 *Id.*

by the statute. In the relatively straightforward case of students using copyrighted code in the classroom setting, the use is fair. However, once the nature of that use changes, so does the copyright analysis. If those same students were to use the copyrighted code outside of the classroom and, for example, develop a module and post it on the Internet, such use is likely to be an infringement of the original copyright.

On the other hand, if the use of the copyrighted work falls outside of the express permissible purposes, the developer can tailor her use to navigate the fair use factors in her benefit. First, the purpose and character of the developer's use – the developer should steer clear of commercial uses (court's are not likely to allow a developer to profit from the unauthorized use of another's work) and instead try to transform the original work by adding new expression and adding value to the original work by creating something new. Second, the nature of the copyrighted work – the developer will have a strong case of fair use if the original work contains code that is not subject to copyright such as, functional or utilitarian code, scenes a faire code, or public domain code. Third, the amount and substantiality of the portion taken - the less the developer takes, the stronger her case for fair use will be. However, even if she takes only a small portion of a work, her use may not be fair, if the portion taken is the heart of the work, i.e., the few lines of code that really make the program the program. Fourth, the effect upon the potential market for the original – if the developer's use deprives the original owner of income or undermines a new or potential market for the original code, such use will severely weaken her case for fair use.

Copying Code Not Protected By Copyright

A prima facie cause of action for copyright infringement requires that the plaintiff prove that protected elements of its work have been copied.[137] In other words, if the code copied is not protected by copyright, there is no copyright infringement. Accordingly, a developer may use the unprotectable elements of a computer program, without permission from the upstream developer, and the subsequent developer will not infringe the copyright protecting that particular program. The salient question then becomes, which elements of a computer program are protected by copyright and which are not?

In the previous section, we discussed several examples of parts of programs that may not be protectable. Recall that the purpose or function – the idea – of a utilitarian work – like computer software – is not protected by copyright. Similarly, factual data is not protectible, despite the fact that the software program, as a whole, may be protected by copyright. Code in the public domain and code falling within the scènes à faire doctrine are also not protected by copyright.

A developer is free to use any elements of an existing program that are dictated by external factors such as efficiency, compatibility and interoperability requirements, computer manufacturer design standards, hardware and software specifications, widely accepted target industry practices, and widely accepted programming industry practices. A developer is also free to use any elements of an existing program that have entered the public domain. Finally, a developer is also free to use elements of an existing program that are standard, stock or common to a particular subject matter under the doctrine of scènes à faire.

137 52 Am. Jur. Proof of Facts 3d 107.

Copying De Minimis Lines of Code

De minimis non curat lex. Roughly translated to English, this legal maxim means the law does not concern itself with trifles. Applied to copyright law, this maxim means that copying which has occurred to such a trivial extent does not constitute an actionable claim of infringement. In other words, a developer who uses a small amount of code, without permission from the upstream owner, does not infringe the copyright protecting the upstream program. For example, one court held that copying thirty characters from approximately fifty pages of source code was *de minimis*, and not an infringement.[138]

However, a developer should be aware that courts do not approach the *de minimis* inquiry in a vacuum; but rather, will consider the context in which the copying took place. As the great Judge Learned Hand said, "no plagiarist can excuse the wrong by showing how much of his work he did not pirate."[139] The court will measure the quantity of the portion used, but it will also measure the quality of portion used. The analysis will occur at the module level and even if only a small amount of code has been used, a court is likely to find it an infringing use if the portion used constitutes the heart of the original work. This last point should ring familiar. The *de minimis* inquiry is part and pacel of the third prong of the fair use analysis – the amount and substantiality of the portion used in relation to the copyrighted work as a whole.

Testing For Derivation

The term *derivative work* is paramount within the open source software community. Derivative works are part and parcel of open source software projects – by its very definition, open source software participants are encouraged to modify, recast, transform, and adapt the source code and redistribute it back to the community for further modification, recasting, transformation, and adaptation.

Remember that the Act defines a derivative work as is a work based in whole, or in substantial part, upon a pre-existing work, and recasts, transforms, or adapts the underlying work in some way.[140] For copyright protection to extend to a derivative work, “the additional matter injected in a prior work, or the manner of rearranging or otherwise transforming a prior work, must constitute more than a minimal contribution.”[141] In addition requiring more than a minimal contribution, a derivative works requires permission from the owner or owners of the copyright in the underlying work. In the open source software community, this permission typically comes in the form of an open source software license. The question becomes, then, when does a developer create a derivative work? The answer, as one might imagine, is not entirely clear.

At one end of the spectrum, a new program will be held to be a derivative work when the source code of the original program was used, modified, translated or otherwise changed in any way to create the new program. A developer can avoid copyright infringement in this instance and her work can be deemed to be an authorized derivative work with one more requirement. She must

138 *Vault Corp. v. Quaid Software Ltd.*, 847 F.2d 255 (5th Cir. 1988).
139 *Sheldon v. Metro–Goldwyn Pictures Corporation*, 81 F.2d 49 (2d Cir. 1936).
140 17 U.S.C. § 101.
141 NIMMER, *supra* note 20.

have the permission from the upstream copyright owner to create that derivative work, and that means remaining within the bounds of the original program's license. Conversely, a work will not be held to be a derivative work where the developer merely applies minor, trivial variations to the source code of the original program without adding anything original of her own. The resulting work will constitute a non-literal copy of the original program, thus infringing the copyright of the original program.

At the other end of the spectrum, a work will not be held to be a derivative work where the developer uses library functions and other off-the-shelf routines contained in an original program, without ever touching the original program's source code. As we have seen above, these components of the original code are not copyrightable themselves, given their highly functional nature, and as such, downstream developers are free to use them in their subsequent works. This work is an independently created new work, not a derivative work.

Utilizing Free and Open Source Code Licensed By Others

An open source software license is "merely a mechanism by which the copyright owners place limitations on the downstream end user's ability to utilize the software code [under the Copyright Act]."[142] In closing, this article is not intended to inform developers about black letter compliance with the open source software license under which they are operating. Rather, it is intended to suggest what a developer is and is not permitted to do in the grey areas outside of the open source software license under which they are operating.

By adhering to the principles illustrated by this article, a developer may utilize free and open source software code and steer clear of the thickets of copyright infringement. The four factors of the Fair Use doctrine stands ready to provide a developer with a safe harbour for her use preexisting open source software code or if the purpose of her use is "criticism, comment, news reporting, teaching, scholarship, or research."[143] A developer can utilize the unprotected elements of preexisting open source software code in her program, without creating a derivative work of the original program. Likewise, a developer is free to utilize those elements of preexisting open source software code that are in the public domain. Although the code she develops is a derivative work, she will not infringe the copyright protecting the preexisting program because her work is one that is derived from the public domain elements. A developer can also utilize a small, *de minimis*, amount of open source software code, so long as the code she uses does not constitute the "heart" of the preexisting program. Finally, a developer can simply create a new, original program, and not a derivative, by utilizing any combination of the above unprotected elements of a preexisting program, or by changing the preexisting program so much that the new program differs substantially from the original.

About the authors

Omar Johnny *is a legal extern at Nielsen Company in the Corporate Legal Department working*

142 Brad Frazer, *Open Source is Not Public Domain: Evolving Licensing Philosophies*, 45 IDAHO L. REV. 349, 365 (2009).
143 17 U.S.C. § 107.

on patent law. Johnny is a second year law student at New York Law School. Johnny holds a B.S. in Computer Science from Hobart College.

***Marc Miller** is a third-year student at New York Law School where he is a Student Research Fellow at the Institute for Information Law & Policy. Miller is also affiliated with the Institute as a John Marshall Harlan Scholar and is a Notes & Comments Editor of the New York Law School Law Review. Miller holds a Bachelor of Science in Management Information Systems from the University of Vermont.*

***Mark Webbink** is a Visiting Professor of Law and Executive Director of the Center for Patent Innovations at New York Law School. Webbink is also a Senior Lecturing Fellow at Duke Law School and has served as an Adjunct Professor at NCCU Law School. From 2000 to 2007 Webbink served in various capacities with Red Hat, Inc., including General Counsel, Deputy General Counsel for Intellectual Property, Senior Vice President and Secretary. Webbink presently serves on the board of directors of the Software Freedom Law Center and on the advisory board of the Axial Exchange. Webbink has written and spoken extensively on the subjects of open source software, software patents, and patent reform. Webbink received his B.A. Degree from Purdue University in 1972, his Masters in Pubic Administration from the University of North Carolina – Chapel Hill in 1974, and his J.D., magna cum laude, from North Carolina Central University School of Law in 1994. Webbink maintains a website on open source and intellectual property law at www.walkingwithelephants.com.*

Package Review as a Part of Free and Open Source Software Compliance

Martin von Willebrand[a], Mikko-Pekka Partanen[b]

(a) Attorney, Partner, HH Partners, Attorneys-at-law, Ltd; Chairman, Validos ry; (b) Attorney, HH Partners, Attorneys-at-law, Ltd; Open Source Specialist, Validos ry.

DOI: 10.5033/ifosslr.v2i1.37

Abstract
Free and open source software ("FOSS") package review is an essential part of license compliance when businesses take into use FOSS. This article discusses the practical process of package review and the legal questions that arise and conclusions that can be made. Furthermore this article presents the process and a number of legal conclusions applied by Validos ry, an association for performing package review and sharing its results. The purpose of presenting a particular process is to share and improve the applied methodology with a long-term vision of unifying the expectations for package review and license appraisal, thus contributing to the ease of taking into use of FOSS by businesses.

Keywords
Law; Information Technology; Copyright; Licensing; Free and Open Source Software; Compliance; Sharing

Info
This item is part of the **Articles** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

1. Introduction

Free and open source software compliance processes aim to enable compliant use of Free and Open Source Software ("FOSS") packages. This article addresses a part of the FOSS compliance process from a perspective of a company and also, to some extent, by a group of companies. The part addressed is package review. In order to achieve compliant use of FOSS, the package review process must identify and record the correct package, identify and record all applicable licenses and their obligations and to some extent copyright holders, identify eventual license incompatibilities and report all of this information in a manner that allows compliant use – even correction of incompliances – in the needed use scenarios. This article's viewpoint to these objectives, is the process through which the objectives can be achieved: the process to identify a package (section 2), the process to inspect such package and its licensing (section 3), the legal conclusions used in appraisal of licensing (section 4), reporting and storing or even sharing the results of the review (section 5) and suggesting corrective measures for non-compliant packages (section 6).

Open source compliance is a wider question to which the package review process belongs. For both a strategical and organizational view on open source compliance, an overview with practical examples is presented by Richard Kemp in his article Towards Free/Libre Open Source Software ("FLOSS") Governance in the Organisation.[1]

1.1. Purpose of this Article

The writers of this article perform compliance work for Validos,[2] an association established for performing compliance review work and sharing the results between all participating companies. The compliance review reports created by Validos are stored in a joint and growing database in a manner that enables reuse by all member organisations,[3]

The purpose of this article is not only scholar, but also practical. This article presents package compliance review processes used by Validos[4] for the purpose of sharing information and also opening up a documented compliance process for criticism and therefore improvement. With criticism and improvement suggestions, this article can be developed into a robust and practical guide on legal package review in open source compliance. We invite all readers to participate into such development.

This article is provided with a CC-BY-SA license that allows derivatives of the article. Thus, elements of this article may be used in creation of individual compliance review instructions.

1.2. Scope and limitations of the Article

A package, as used herein, means typically a single identifiable file that is offered for download by

1 KEMP, R.. Towards Free/Libre Open Source Software ("FLOSS") Governance in the Organisation. International Free and Open Source Software Law Review, North America, 1, dec. 2009. Available at: http://www.ifosslr.org/ifosslr/article/view/19/51. DOI: 10.5033/ifosslr.v1i2.19.

2 Validos ry (http://www.validos.org) is an association based in Finland with 12 member companies representing a turnover of over EUR 1 Billion.

3 Validos shares all reports on open source packages between all members. The basic logic is that each member participates on at least a certain level (depending on the member revenue) and that the results of all of the work is shared via an extranet. An important element in Validos is that the review process and the reporting has been geared to support reuse in different use scenarios – at the same time this means that members still have need for member-specific decisions on, *e.g.*, linking questions in relation to member's proprietary software. The Validos database grows via the requests for review by members. Currently the database holds reports on more than 200 FOSS packages, and approximately 4 new packages are added per week. The reuse rate of new review requests varies between 0 - 75 %, meaning that at best 75 % of the packages in a review requests can be obtained from the database with no new work required.

4 However, even if most of the process description reflect Validos processes, this is not an exact description as the process is continuously developed and also member preferences affect the compliance work of individual packages.

an open source project. A package may come in tar.gz, zip or other compressed format.[5] As a compressed format, a package may include any number of files and subdirectories, as determined by the project offering such download.

Package review is not limited to pure FOSS packages. This is due to practical reasons: packages tend to contain files or subdirectories or other elements that are not open source, according to definitions by the The Open Source Initiative (OSI) or the Free Software Foundation.[6] Although some files may not be pure FOSS, their use might anyhow be relatively unrestricted from a corporate perspective, and therefore, their use is often controlled by the same process as the use of FOSS. From a company's perspective the compliance process can be similar or same regarding all software packages that can be obtained without charge from internet sources, such as FOSS, public domain software or freeware.

An open source project can be either a group of individuals, a separate legal entity or a part of the activity of an existing company, or even a mixture of these. In this article, we use the term "open source project" to describe these. We refrain from analysing eventual differences resulting from the type organization of the project. However, compliance review is done in relation to the copyright holder and eventual license grant by the copyright holder.

Package compliance review results in information that is generic and may be used by many companies and may also result in information that is specific to a use case, and as such may not be used as well by others. Since the generic review results are useful to many, its creation can be done in a collaborative fashion. However, reuse by many poses requirements on the review process. One perspective in this article is satisfying such requirements and enabling sharing of the generic elements of review results. Also, this article concentrates on the generic elements of the review process and not on the specific questions, such as linking with member-specific software.

This article aims not to discuss eventual risks in non-compliant use of FOSS packages: it aims to discuss a part of the process for ensuring compliant use of FOSS packages. Further, we do not intend to thoroughly or orderly discuss methods of analysing and assessing risks in relation to use of FOSS packages or licensing uncertainties, although some parts of the article touch risk appraising questions.

This article only discusses use of FOSS packages by companies in relation to redistribution of the FOSS package by the company, and not other use scenarios (such as use for a commercial service or internal use). The boundaries of redistribution are not discussed. This article does not discuss possible liability questions in collaborative production of compliance information.

We have refrained from analysing compliance review questions from a perspective of any single jurisdiction. Traditional legal sources do not address the practical questions of package compliance review: at least we have not found such information from any jurisdiction. At the same time, the companies need to apply the review results in multiple jurisdictions in a unified manner or at least with only small variations. In defining the legal conclusions we present here, we have assumed

5 For example Unix source code of Apache http server version 2.2.15 is distributed in package httpd-2.2.15.tar.gz.

6 The definition of open source by the Open Source Initiative: http://www.opensource.org/docs/osd (retrieved on 4 May 2010) and the definition of free software by the Free Software Foundation: http://www.gnu.org/philosophy/free-sw.html (retrieved on 4 May 2010).

that the general principles of copyright and norms in interpretation of licensing or similar texts, are similar in most jurisdictions. To the extent this is not so in relation to the legal conclusions we present, we would be delighted to be corrected. The writers of this article are Finnish lawyers and thus come from a European continental, and more specifically Nordic, background.

1.3. Methodology

From a perspective of legal methodology, what is the value of publicising a package review custom, even a developing one? We have noted that current legal literature does not much address very practical questions that need to be addressed in package review. Also, we see it highly improbable that such legal conclusions that we have come into would be determined by an authoritative source, such as a court, any time soon. Taken into account the amount of jurisdictions and the amount of different conclusions, this is evident. Thus, in lack of a way to establish a correct package review methodology by investigating traditional legal sources, we have decided to strive for the development of a consensus by the legal community interested in FOSS and thereby also the wider legal community engaged with open source projects. If such a consensus would be established, the practical risk of non-compliance would be lower, since if a company adhered to the consensus, the probability that a right holder in an open source project would require a company to interpret licensing differently, would be lower.

1.4. Introduction Validos FOSS Review Process and the Legal Conclusions Used Therein

The Validos process returns a compliance value for a package which is quite simple: a package is found to be (i) compliant or valid, (ii) possibly incompliant or (iii) incompliant /containing clear risks.[7] Compliant means that the licensing of a package was found clear and no incompatibilities within the package were found. Evaluation against the redistribution license of the member is out of scope (a member-specific question that is not part of a generic compliance process: that information has little value for reuse in other use situations). In addition, one outcome of the process is the use instructions for redistributing the package and other reports (section 5) and possibilities to correct found possible or clear non-compliances (section 6).

FOSS packages are often licensed in ways that are not clear or unambiguous. The process described in this article has been used within Validos on more than 200 packages containing thousands of sub-packages. Of those packages 65% have been found fully compliant in accordance with the process.[8]

This percentage includes a set of legal conclusions applied: we have deemed that certain typical situations are considered compliant, as long as defined criteria are fulfilled and contrary

7 In Validos, each package receives a value “Compliant”, “Possible Incompliance”, “Clear Incompliance” for every standard use case (redistribution, commercial service, development tool and internal use). This is viewable as one line information. We also use a marker to signify that decisions are required (*e.g.*, a GPL-licensed package can be fully compliant but will anyhow require a decision regarding linking by the user of the package). The first level report will then offer a risk pointer with short explanation of the reason for the given value, *i.e.,* “Why was this package tagged with Possible Incompliance?” and will also detail the files and folders affected. The second level report expands the first level report with all the details of the review process.

8 The percentage does not reflect the gravity of the non-compliance: a single file in a 25,000 file package may create a (possible) non-compliant tag: it is the purpose of the use instructions to reflect the gravity and point to the risk in question. It can also be very easy to correct a non-compliance.

indications are not found. The legal conclusions applied are presented and discussed in section 4.

Legal conclusions could be also made on an *in-casu* basis. However, due to the frequent occurrence of most of the situations meant in the conclusions, we assert that most companies would do well to form a policy on these questions. Leaving every question to be individually appraised by an open source review board will not result in better compliance decisions, but rather ineffective working methods and unnecessary variation.

It is to be noted that the approach described aims to enable companies to utilize free and open source software. Instead of an approach to enable companies to use open source software, a company could adopt a very risk averse approach, and not, *e.g.*, use any files which do not contain a clear license header due to perceived licensing ambiguities. This would lead into non-use or a very limited use of open source packages by such company. Thus, we acknowledge that not every company may be willing to accept these conclusions as a basis for their compliance work.

A number of questions will still need to be answered separately from the conclusions described in this article. Typically these questions are submitted to an open source review board, or similar ad-hoc formation including an engineer, a process controller and an open source knowledgeable lawyer. These situations include:

1. Situations in which the legal conclusions do not apply, *e.g.*, when the process finds a or possibly incompliant package; and
2. use-case specific decisions, such as questions on interaction with proprietary software (*e.g.*, linking).

2. Identifying a Package

Objective: When a compliance review request is received the first compliance task is to identify the package described in the request. Correct identification of the package ensures that the compliance work is done for the correct package and that the information can be later reused by others: others need to be able to match the reviewed package with the package they are planning to use.

Description: Typically an open source project has different versions of the packages it offers, available perhaps on different hosting sites, project pages or source code management systems. This can lead to uncertainty regarding which is the right software package to review, especially in situations where the review is performed by a specialised unit or it is performed much later than the actual software has been taken into use by a company. To avoid such uncertainty and correctly establish the package to be reviewed, the reviewers should note at least the following different versions:

Different Versions / Uncertainties	**Comment**
Wrong Project	Sometimes another open source project may have a confusingly similar name, or there might be an old, out-dated and no-longer used web-page of the same project.
Sub-Projects	An open source project may be divided into multiple sub-projects
Larger Packages v. Smaller Packages	Packages may be offered in versions which include different combinations of the software of the project or third party software. A typical example is a version which includes all required dependencies compared to a version with only code created by the project. Also, sometimes projects provide versions which do not include certain code (*e.g.*, a patented cipher).
Platform Variation	Packages may differ for different platforms (such as Debian, Red Hat, Windows etc.)
Binary v. Source	Packages may come in versions including only source code or only binaries or both. Sometimes the source version is more encompassing than the binary version (*e.g.*, source is provided for all platforms) or vica-versa (*e.g.*, the source is only provided for the code created by the project and dependencies are only in binary format).
Development Versions	Finally, most projects have different development versions, *e.g.*, versions running from 0.1 to 2.72. Development versions can also be indicated with letters 1.0a etc.

Table 1: Possible uncertainties

Recommended process: Review requests should contain at least the project name, its web page and the development version of the package. File name and URL of the package to be used might not be sufficient, as the user might refer a binary only package and review should be performed on source version or both the source and binary version. This is why the project name and the web page of the project are always needed.

Review requests sometimes refer only binary packages: if this is the case, one critical element is to find the source version matching the binary version. If matching is not clear and it cannot be cleared with the unit requesting the review, the remaining option is to inspect source files and compile the binary to be used from the reviewed source. In this case, only the source version is reviewed.

In practical terms, the information in the review request will need to be assessed against the information provided by the project, in order to establish the correct package to be inspected. Possible discrepancies need to be solved with the company/unit/project requesting review of the package. If request includes partial information, eventually completed information should be sent back to the requesting unit for verification.

As a part of identifying the package, the inspected file should be stored and a unique signum for the file should be created (such as an MD5-signum). At best, the unique signum is provided by the project and that signum can be checked against the signum created for the stored file. If the projects use a signum, the same signum type can be used for the stored file. In addition, the preferred signum type should be created in any case so that the stored files have one unified signum for each file.

3. Review of Package

3.1. Collect Information

Objective: After a package has been identified in accordance with the previous section, the actual review of the package begins. The objective of the inspection of a package is to collect all information that is relevant for the compliant use of the package and to analyse arising legal questions.

Recommended process: The inspection starts with manual inspection of the web pages of the project and the downloading of the identified package. The web pages are also inspected for license information and any related material such as statements regarding known patent issues or export restrictions. Occasionally additional copyright, license or author information needs to be searched via search engines from other public sources such as related mailing lists. Found license information is recorded for archival purposes by taking a screenshot or printing an electronic copy.

When the package is downloaded and archives extracted, the package is briefly inspected to form an overview of the included folders, documents and libraries. Validos process also includes uploading the package to FOSSology source code analysis software after the package has been downloaded. FOSSology is an open source licensed tool that can be used to analyse source code. At the time, the most useful feature of the software is its ability to find license text matches from the source code of a package. This is done remarkably well as the software identifies reliably also license fragments and modified license texts.[9] We have found that the amount of licenses not found is very low.[10]

There are also other source code analysis software tools, which can be used in open source compliance processes.[11] Sometimes it is necessary to use text search tools such as grep to find and collect copyright and license notices from large code bases.

The practical review of Fossology results can be performed as follows:

1. .Overview of the Fossology Results. When Fossology has processed the uploaded software package, it displays result of the check as a list of found matches. The results should be briefly examined for the purpose of forming an overview of the included licenses and phrase matches.

9 For more information, see http://fossology.org.

10 The current road-map of FOSSology includes the reporting of copyright notices found in packages, to be released in version 1.2, probably very soon. This feature will add to the review certainty, as unidentified licenses can be picked up by their copyright notices.

11 Most known alternatives are different offerings by Black Duck, Palamida and OpenLogic. There is also another free tool, OSLC – or Open Source License Checker (http://sourceforge.net/projects/oslc/, link retrieved 2 May 2010), which is not much developed currently.

2. Review of Phrases. If Fossology finds suspicious text matches that do not correspond with any known license text, it can point them out as phrases. As these findings can potentially refer to proprietary type licenses or other restrictions, the matches need to be reviewed. This is done by reviewing the preview view for the match "phrase", or if necessary, by reviewing each file that contains the spotted phrases.[12]
3. Review of License Text Matches. Fossology scans the uploaded file against license texts in its knowledge base. However, the listing shows only textual matches and not legal matches: e.g., a file with a dual license will probably be shown as a hit with two different licenses.. Manual review of the results is therefore required with help of the interface provided by Fossology. The findings should also be reviewed to check that the matches corresponds fully with the stated licenses and eventual license modifications are found. Most projects luckily employ similar statements so that each file does not need to be inspected separately.[13]

3.2 Analyse Collected Information

Objective: After collecting data from project web pages, documentation and source code it must be analysed. The objective is to identify the level of clarity of licensing and eventual incompliances and other issues.

Recommend process: At Validos, we review at least:

1) Main License Clarity

How strong and reliable is the information on the license applied by the project (we refer to this license as the main license). According to our experience it is not always clear what the main license is. Typically these situations are related to contradictory or incomplete license statements in project web pages and downloadable packages.[14] Sometimes these can be solved satisfactorily by finding a common nominator in the package: (*e.g.*, unclear references to GPL 2 and GPL 3 licenses on the webpage can be solved, if the package thoroughly refers to "GPL 2 or later"). However, in many cases solving in-clarity regarding a main license requires contacting the open source project or the relevant author. If the reviewer needs to make a judgement on the license, the package receives a "possible incompliance" tag from the review process.

2) Compliance of Existing Sub-Packages or Sub-Components

One review item is to find and list the existing sub-projects or third party projects and the

12 Preview view of Fossology displays a one or two row excerpt of found phrase, in many cases this is enough to determine whether the found match is relevant in a licensing sense, or refers to non-compliant license. If a preview is not enough to resolve whether or not there is really any issues, the files source code can be accessed from the tool reviewed in detail.

13 For example the tool can show that inspected package contains 10 000 files with match "'GPL v2'-style ", while it is not effective to check each of 10 000 match for unexpected modification, at least some source files should be reviewed. Should any inconsistencies be found, the findings should be reviewed in detail. A feature that separates different types of matches would make this process faster.

14 For example, situation where a project web page contains a statement "Licensed under the GPL." where word GPL contains a link to Free Software Foundation's GPL license page, which nowadays contains the version 3 of the license. At the same time a download package can contain a statement "Licensed under the GPL v.2 only".

applicable licenses (sub-licenses).

Typically FOSS packages include code created by others than the main copyright holder. While code reuse is one of the driving forces of open source development, it is also a common source of legal risks. This is caused by the sheer number of licenses (whether open source or more limited licenses) that are not compatible with other licenses, which combined with the fact that developers tend to be more interested of coding than licensing, causes often situations where some included sub-licensed files or components are not compatible with the main license.[15] Therefore, one of the main tasks of a package review process is to point-out any situations where all license requirements cannot be fulfilled simultaneously when the software is distributed. Equally important is to find files that may not be distributed at all, such as components licensed only for evaluation use. License compatibility checks are done by reviewing stated license information and results of source code analysis. If clearly or possibly incompliant licenses are found, corresponding components are reported with necessary detail, usually at folder or file level and the report summaries receive a corresponding value. Additionally, when discrepancies have been found, corrective measures, which can be used to solve the issue or mitigate risks caused by problem, can be suggested.

3) Other Elements such as Patent and Export Control Related Information

As a note, information that relates to patents or eventual export control related questions, can also be collected.

Occasionally, license problems can be solved by contacting open source projects for clarifications. We have found this approach to be welcomed by projects and in many cases projects correct or clarify issues not only in their replies but also clarified the information provided by the project. We see contacting of projects as a way of contributing back to free and open source projects. The findings of the review process should always be recorded in a format enabling quality control, sharing and reuse.

4. Legal Conclusions in Appraisal of Licensing

As we mentioned above, the Validos process returns a compliance value for a package which is quite simple: a package is found to be (i) compliant or valid, (ii) possibly incompliant or (iii) incompliant /caining clear risks. In addition, one outcome of the process is the use instructions for redistributing the package (section 5 below) and possibilities to correct found possible or clear non-compliances (section 6 below). However, FOSS packages are often licensed in ways that are not clear or unambiguous.

15 In pure package review, as the one described in this article, the incompatibility is assessed within a package or a combination of packages. The question of license compatibility in relation to proprietary or other software of the member organisation is not a part of generic package review. That question becomes member-specific and cannot therefore be shared with other members (the answer to a member-specific question also has little value to others, or little value for reuse in general).

4.1. Files with No License Headers

Issue: Packages contain files with no license headers. Under which license should these files be considered to be licensed?

Conclusion: Files with no license headers are considered to be licensed with the closest main license, as long as there are no other indications. *e.g.,* a library or folder within a package may contain an open source license and 10 files of which the most important one contains a license header and the rest do not have any license header. In this case all the other files are considered to be licensed with the "main license" of that folder, unless there are contrary indications.

Typical Contrary Indications: A copyright notice by a third party that differs from the copyright notice of the rest of the package and there is no indication of that party participating in the same open source project. Statements on proprietary licensing, such as "Copyright ATT 1989. Proprietary and unpublished".

Discussion and arguments: This is the widest question in package review. Most of the packages include files with no license headers. It can be envisaged that bigger projects will embrace detailed policies and licensing practices which solves this question at the source,[16] but the amount of projects will continue to increase and this issue will persist. Companies taking into use FOSS packages will need to resolve this question somehow. Small to medium size projects mostly do not see this as a problem and in lack of a unified approach to offer to projects, companies will mostly need to resort to policy decisions on this.

The conclusion we propose seems to offer a practical solution to this question. The weakness of the argument is that the files do not contain any license headers and the conclusion seems arbitrary. However, it must be noted that there is no widely accepted instruction or practice to include a license notice in each file and one could also ask why not include a notice for each line of code. A notice can equally be placed on the folder level, as it can be placed on a file or package level. Furthermore, we are not aware of any legal obligation in Finland or elsewhere to include license notices on a particular granularity, such as at a file level. We deem this conclusion to reflect most authors' intent taken into account the practice in placing license notices. On the other hand, contrary indications need to be reviewed (see above).

4.2. Modifications to Files

Issue: Files may at times contain notices that they have been modified by another party than the original creator. In many of these cases, there is no license reference in relation to the modification. However, the file may contain the original license notice or references of the original author. Under what license should the modifications be considered to be done?

Legal Conclusion: Modifications to files are considered to be under the same license as the rest of the file, unless otherwise is indicated.

Contrary Indications: In most cases, only a reference to a second license or a statement on other

16 Such as the SPDX initiative hosted by Fossbazaar, a working group of the Linux Foundation (https://fossbazaar.org/content/fossbazaar-face-face-meeting-lf-collaboration-summit, retrieved on 4 May 2010)

type of license (such as a statement referring to proprietary type of licensing) is a contrary indication.

Discussion and Arguments: Since the author of the modifications has not expressly stated a license, it can be asked how is his intent to license the modifications expressed. When reviewing individual files this question may be affected also by how the statement of modifications is formulated and how it is placed in the file, in relation to the existing license reference. Possibilities include placing the modification statement and eventual copyright notice before or after the existing license notice, to include it into the existing license notice or to state it as a comment later on in the file. However, in each of these cases, we conclude that the intent of the author is expressed by the fact that he did not add another notice or reference to another notice. In fact, this same argument applies even if the file does not contain notice of its own, but rather a main license is applied (see legal conclusion on Files with No License Headers).

4.3. Licenses Do Not Automatically Change or Automatically Attach

Issue: Many times an open source package that includes GPL licensed files includes also files with other licenses such as MIT and BSD. The GPL license (both in version 2 and 3) requires that a whole is licensed under the GPL license. *E.g.,* MIT-licensed files are considered GPL compliant since it is possible to fulfil both the requirements of the MIT license and the GPL license, at the same time. However, the practice with open source packages is that licenses are not changed or added onto each other, in the file headers. This means that a MIT licensed file within a GPL package continues to contain only the reference to MIT license, and open source projects and their redistributors, do not add GPL license references to these MIT licensed files. The question is whether the license of the MIT-licensed file has changed into MIT+GPL due to the inclusion of the MIT file to the package containing GPL-files. This question has relevance *e.g.,* in cases where a company wishes to use only the MIT-licensed files and wishes to remove the GPL-files. Are the files still licensed with just MIT or should they be treated to be licensed with both MIT and GPL? Does the license change automatically from MIT to MIT+GPL? Or in case the package contains internal incompatibilities, such as Mozilla Public License files and GPL-files forming a whole in copyleft sense: can such incompatibility be rectified by removing the GPL-files?

Conclusion: Files are considered licensed with the information contained in the file, to the extent there is no information to the contrary. The existence a GPL-file in the same package is not contrary information. Licenses of files are considered not to have changed (or not to change automatically) when the whole package is licensed with another license or contains files licensed with another package, even if the license of the file would allow addition of new conditions or new license.

Contrary Indications: Additions to license headers to support an imposed additional license by the project or the redistributor.

Discussion and arguments: It would require interpretation to deem a file containing one clear license statement to be considered licensed with the stated license and another license, in a cumulative manner. However, in case a MIT-licensed file is contained within a GPL whole, it could be argued that the GPL redistribution requirement (copyleft) implies that the MIT-licensed file has been redistributed under the GPL and under the original MIT license (both licenses' requirements applying to the same file on subsequent redistribution). This is not reflected in practice in any way: we have not seen any license headers with license additions of this type. However, even if relicensing MIT-files without the GPL addon, could theoretically be considered to be incompliant relicensing of the GPL-parts, it is not very probable that the right holder of the GPL-part would be interested in enforcing the GPL against this type of behaviour since there is hardly any interest in doing so and this type or relicensing is very common. If the GPL-elements are removed from such a package, we would deem it very strange, if the right holder of the GPL-parts could thereafter exercise control over the relicensing terms of the elements that originally were by another party and under another license. This argument applies regardless, if the elements were licensed with a GPL compliant license, such as MIT or a GPL-incompliant license, such as the MPL. Thus the conclusion is that GPL-elements can be removed from a package to allow *e.g.,* linking with GPL-incompatible packages and also, as the conclusion is founded on licenses not attaching automatically, also package incompatibilities can be fixed by removing elements that cause incompatibilities, at least in cases where incompatibilities are caused by GPL licensed elements.

4.4. Software Copyleft in Relation to Firmware

Issue: Firmware files are at times distributed together with non-firmware software with the intention that the firmware files are run on a separate device and the software is intended for running on a computer processor. May firmware files form a whole, in GPL sense, with software intended to be run on a computer processor, outside of the device containing the firmware? Is there a possible incompliance question in cases where GPL-software running on a computer processor interacts with proprietary licensed firmware?

Conclusion: Firmware, which is intended to be placed on hardware, is separate from a software intended to be run on a computer processor. As such it does not form a derivative of software intended for running on a computer processor.

Contrary Indications: A clear statement by the right holder or licensor of the GPL-licensed software. Even this indication does not cause a clear incompliance, but rather a possible incompliance, since it can also be argued that an attempt by a GPL-licensor to control redistribution of firmware elements, is not effective in a copyright sense.

Discussion and arguments: These series of instructions (firmware v. traditional software) are distinctly separate. The question of firmware files containing mixed code (GPL and proprietary) within the firmware is outside the scope of this legal conclusion.

4.5. Autoconf and Other Build Tools

Issue: Build or similar tools that are licensed with a GPL license are widely used and included in open source packages. The question is whether the copyleft obligation contained in the GPL license should be considered to form a whole (as meant in the GPL) with the rest of the files in the same package. In most cases the rest of the files are also output of such tools, *i.e.,* built with such tools. This question applies to GNU libtools and GNU autoconf tools and Bison parser files

Conclusion: GNU libtools and GNU autoconf tools (and Bison parser files), when contained in packages, are assumed to be used as build tools, unless there is indication to the contrary. GPL-licensing of build tools is considered not to pose requirements to the license, as regards distribution of the rest of the software built with those tools, even if the tools are contained in the same package.

When a file contains the autoconf-exception.[17] the exception is applied, if there exists, in the same package, a file that states "generated by autoconf"[18] (it is not necessary to check whether the file actually is generated by autoconf, the statement is enough).

The Bison exception,[19] if it exists, is applied if there are files that state "made by GNU Bison" and the version of Bison 1.24 or higher. While the wording of the exception sometimes refers only to "use", it is concluded that it means to allow all exploitation rights granted by copyright (copying, modification and publication).

Contrary Indications: Typical contrary indications are other GPL-licensed libraries included in the package and the output of the build process. Also, if the software package would be build tool in itself, then this would be a contrary indication.

Discussion and arguments: Build process can be considered legally as copying of the source code and other elements into a slightly different format as object code and other code organised for execution by a computer. Object code could be considered as modified version and as such a derivative, but as it it a mechanical process that does not normally include human creativity, we would consider the object code to be a copy of the source code. Similarly other parts copied in the process are copies. A similar copy would be an analogue piece of music as a digitized copy. Although the build tools may be and probably are works of authorship, the same applies to build tools as any other computer software: output obtained by using them is not subject to the copyright of the computer software (unless elements are directly copied, which is a contrary indication). In some cases, the instructions given to the build tool could be considered creative, but this is similar to other code given for the build tools for processing, such as the source code. Thus instructions, and files containing instructions can be treated similarly as the source code. In the end, the build tool's license does not affect the license of the code processed by the build tool.

4.6. Dual License

Issue: Many open source packages refer to "dual licensed" files or packages. Many times the wording "dual licensed" is explained to mean that the licensee may choose either of the stated licenses, but also others expressions exists, *e.g.*, "dual licensed with MIT and GPL" or licensed with "CDDL+GPL" with references to "dual license".[20] These latter could be interpreted to mean

17 For example: "# As a special exception to the GNU General Public License, if you distribute this file as part of a program that contains a configuration script generated by Autoconf, you may include it under the same distribution terms that you use for the rest of that program."

18 For example, a file named "configure", which contains text "# Guess values for system-dependent variables and create Makefiles. Generated by Autoconf 2.52."

19 For example, " /* As a special exception, when this file is copied by Bison into a Bison output file, you may use that output file without restriction. This special exception was added by the Free Software Foundation in version 1.24 of Bison. */"

20 For example, see http://wiki.java.net/bin/view/Projects/GlassFishCodeDependencies (retrieved on 2 May 2010)

that both of the licenses need to be applied. Sometimes several licenses are referred to with a separation using the word “or”. How should not clear references to “dual license” be interpreted?

Conclusion: We conclude that the wording “dual license” or use of “or” means that the licensee may choose between the licenses offered, unless there is contrary indications.

Contrary Indications: A contrary indication is an explanation of another type of licensing scheme than a pure dual license where the licensee may choose the applicable license.

Discussion and arguments: The statement “dual license” is also sometimes used to refer to an offering, where obtaining a second license requires payment of a license fee (*e.g.,* a proprietary like license with no copyleft obligations instead of a GPL license, in exchange for a license fee). Except for this situation, we feel that every project using some kind of statement of “dual license” means that the licensee may choose between the licenses. Sometimes the dual license choices are also incompatible with each other, such as the Mozilla Public License and the GPL: in these cases, the theoretical assumption of licensing with both licenses, would not be possible due to that the requirements of these licenses cannot be simultaneously satisfied, and thus a project hardly would require such a license scheme from its users.

4.7. Short License References

Issue: It is not uncommon that FOSS packages or files just refer a license, without containing the actual license text. In these situations it is not necessarily clear what is the applicable license that must be complied when the software is redistributed. *E.g.*, many files refer to a MIT-license without clear definition of the MIT-license. Which MIT-license should be applied? This issue does not refer to ambiguity in license version numbers in cases where there are clear license texts and license versions, but rather to licenses which are more varying (mostly MIT and BSD).

Conclusion: If the license text is not provided, the applicable version is that which is provided by the project that has introduced the respective license. If there is no such project or organization, or it is likely that such initial publisher is no longer maintaining the license, the source of the license text is Open Source Initiative’s list of approved licenses. *E.g.*, the MIT-license text, if not otherwise indicated, means the MIT-license text approved by the OSI (www.opensource.org).

Contrary Indications: Any reference by the right holder or the project to another type of license.

Discussion and arguments: This is really a practical assumption, not necessarily a legal conclusion. Still, it quite probably results in a license and license content accepted by the right holder. The license contents in different MIT-license versions are, from a risk assessment perspective, quite similar: all allow copying, modification and redistribution, so any risk would relate to lesser obligations. A right holder requiring remedies based on application of a certain MIT-license not specified by him, might also have difficulties in such claim. Of course, one could theoretically argue that there is no license.

4.8. GPL and LGPL Version Incompleteness

Issue: Many times projects refer (at project pages, root of the package or source files), in an incomplete manner to licenses and do not state the version of the license, or the information is contradictory. Typically this occurs between GPL version 2 and 3 and LGPL versions 2/2.1 and 3. Which license version should be applied? What if the project cannot be contacted and it is inactive?

Conclusion: When there is incomplete information regarding a license's version, a single point (*e.g.,* source file) defining the license version completely is enough, provided there is no conflicting information. If the version is totally unspecified in every place, then the rule on all LGPL and GPL license versions applies: user may choose the version of the given license. If there is no single point that defines the license version, and the project web pages refer to http://www.gnu.org/licenses/gpl.html and the date of the package is earlier than 29 June 2007 and the project is inactive (does not reply to queries), then we consider GPL version 2 to be the correct license.

Contrary Indications: In relation to a single point establishing a license version, any contradictory reference to another version will create a possible risk.

Discussion and Arguments: Regarding references to http://www.gnu.org/licenses/gpl.html for packages earlier than 29 June 2007, it could be said that the project might have earlier referred a completely different license, but this is quite improbable. The best way to solve this question, is to ask the project, but sometimes the project is inactive. Inactivity of the project supports that the project has not intentionally changed its license.

4.9. Source Code as Documentation

Issue: Several licenses require provision of copyright, license and similar notices in the documentation to the end-user. How can this be fulfilled in outbound licensing, *i.e.,* what are the exact requirements of the licenses of the packages to the redistributor? Is it enough that the notices are provided in electronic form and can they be provided as a part of the source code? Is it enough that just source code is provided to the licensee / end-user?

Conclusion: Provision of source code to the licensee / end-user fulfils the requirement to provide the copyright, license and similar notices to the licensee / end-user.

Contrary Indications: Clear text to suggest different method of provisioning the notices.

Arguments and Discussion: Notices are contained in the source code. Typically provisioning of the source code is considered as providing the end-user more than just the notices. Thus, if the redistributor provides the end-user the source code containing the notices, the notices are provided to the end-user. It can also be discussed, whether the source code, when delivered like this, is documentation or not. Source code can also be considered as part of documentation, since it provides detailed information on the functioning of the software, its authors and licensing. Also, separate notice documents are not very useful and tend to become lengthy, uninformative documents, and we are not sure whether the right holders wish such practices to be undertaken. As an additional point, we have not seen any license requirement, which would require non-electronic distribution of documentation.

5. Reporting, Storing and Sharing Review Results

Storage and sharing can be considered parts of reporting, since package reuse requires ability to reuse results of earlier compliance reports. Review results should be stored also for quality control purposes.

Thus, reports on review results have a number of requirements and objectives:

- Reports should be easy to use and (thus) enable compliant use of the package, to the extent possible. The language used should be clear and concise, to enable professionals with different educational background to review the reports.;
- Reports should enable risk assessment in cases packages were found possibly compliant / possibly risky; or even risk assessment of packages found compliant, if a certain legal conclusion was used (in case a user wishes not to accept such legal conclusion);
- Reports should enable variations in risk preferences for different use cases; and
- Reports should enable sharing (separation of generic and use-case specific information).

5.1. License Requirement Simplifications

The Validos process uses simplified license summaries to instruct the redistribution of FOSS packages.

The advantage of this method is that the license requirements become easier to understand, more standardised (same requirement in different licenses is expressed in only one way) and faster to apply. The disadvantage is that the requirement might be applied in a wrong way, since the wording has changed from the original license text. The process needs therefore to provide also the information on the licenses applied, so that the user may read the licenses directly.

However, the writers of this article contend that license requirement simplifications result in a better end result for compliance since full license texts can also be misunderstood. Also we further contend that, even if an open source review board is used for each released project or product, not using simplified license information will result in a non-effective and non-standardised working way. In practice, the compliance officers and lawyers will memorise license requirement simplifications or they will reread license documents. It would therefore be more standardised and effective to use license requirement simplifications reduced to writing in an open source review board too.

5.2. General Use Instructions and Package Specific Use Instructions

The Validos-process has introduced a general use instruction,[21] with the objective to help

21 Link to http://www.validos.org/en/about-validos/37-validoksen-toimintatavat/66-general-use-instruction-for-open-

instructing redistribution of individual packages. The general use instruction covers most frequent requirements in FOSS-licenses. The general use instruction can be applied to all packages: it reduces the length of package specific instructions and standardises the redistribution methodology. Package specific instructions complement the general use instructions with requirements that are not covered by the general use instructions.

The general use instruction of Validos, includes the following:

1. Keep all copyright notices, license references, license texts, notice-texts and warranty disclaimers intact and redistribute these together with the package when you redistribute the software package.
2. Do not use the name or any mark of (i) the software, (ii) the project, (iii) any author or (iv) any copyright holder in any marketing, promotional or similar material or for such purposes, nor in the name of your product or in any other such way.
3. When you modify an open source package and redistribute it as modified, you should always mark your own modifications clearly added with the date of your change. This is typically done by markings at the beginning of the relevant file.
4. When you distribute the open source package as binary, you should also preferably always distribute the source code distribution of the original open source package together with the binary and state in the binary that the original source is distributed together with the binary.
5. If item 4 is not possible (*e.g.,* due to space restrictions) verify that all separate text files listed in item 1 are contained in the binary distribution in a corresponding directory.

The general use instruction covers all the requirements in a number of frequent licenses (such as MIT, BSD and Apache 1.1 with legal conclusion (4.9), Apache 2.0 except patents) and many of the requirements of other licenses. The general use instruction standardises the compliance process for all FOSS projects and makes the instructions for additional license requirements simpler. Therefore a package specific report on a purely GPL 2 licensed package needs to cover only the requirements that go beyond the general use instructions (*i.e.*, copyleft requirement).

Even if many licenses do not require source code redistribution, the item 4 in the general use instruction has been found as a useful way to standardise processes and to reduce work in compiling license notices to separate documents. See also legal conclusion 4.9 on using source code as a documentation.

5.3. Risk Preferences and Assessments

Compliance review will mostly find packages as compliant or possibly compliant. When a package is possibly compliant, a risk assessment is required. Typically it is a question on legal analysis: are these licenses compliant, when combined in this way? Or, is this file licensed with license version 2 or version 3, when the license reference is ambiguous and indications to both license versions exist? Do we need a patent license for a certain cipher even if we remove file x?

The above discussed questions can be solved by policy decisions or further review, such as by

source-packages (retrieved on 2010-04-29) contacting the open source project or research on cipher patents. These actions might still result in not entirely clear answers. This is when risk assessments are required.

Different use cases might have different preferences for risks, costs and time. The preferences may vary depending on the company or may vary depending on the unit within the company or even within different projects within the same unit. The compliance review reports should separate between information and risk assessment so that risk assessments can be done on a use-case specific level. Validos process does this by not doing the risk assessment, just pointing to the risk and explaining it. However, the legal conclusions we have discussed in section 4, can also be seen as risk decisions, although they are very generic. The reports could also include information on which legal conclusions were applied: this would enable policy decisions not to accept certain legal conclusions.

Information that allows risk assessments is not necessarily simple and straightforward. Therefore it might not be suitable for a simple and straightforward reporting of use instructions for FOSS packages. We have addressed this concern by providing only high-level information on a higher level with pointers to more detailed information. The Validos process provides a one-line report on each package using colour coding for different typical use cases, and then, at certain colour codings, a risk pointer in the package specific use instructions. The pointer includes general information on an eventual risk and points to the full report describing the estimated risk in full. The full report not only allows risk assessment, but also quality control.

5.4. Enabling of Sharing

Package compliance review results in information that is generic and may be used by many companies and may also result in information that is specific to a use case, and as such may not be used as well by others. Since the generic review results are useful to many, its creation can be done in a collaborative fashion.

In order for sharing to become possible, two things must happen: 1) the collaborative production of compliance review information must be more effective than production of the same information by each company separately and 2) the information to be shared must not be confidential. The requirement on effect includes that the information must be readily usable within the processes of the user companies and their supply-chains (upstream and downstream). This in turn means that addition of use-case specific information should be possible without sharing that information to others.

The first requirement is fulfilled by the basic fact that there are many user companies of the same open source packages. (*E.g.,* if the Linux kernel is used as a basis of redistributed products and projects by thousands of companies, then it is not effective for each of the companies to do the compliance review separately, if a working joint way of doing the review exists. The same applies each time a new version of the kernel is issued. Even if a joint compliance effort would need to be much more robust, and therefore perhaps multiple times more costly to produce, still the cost per company would be much lower than individual production of the compliance information by each of the companies).

The key to enabling sharing of FOSS compliance review information is to limit the information to generic information that can be obtained from the open source packages. Another important elements is that the use-case specific information must be easy to add to the generic information.

6. Suggesting Corrective Actions for Found Incompliant Packages

It is not uncommon for FOSS packages to contain code that causes them to pose potential or clear risks when redistributing them. However, the fundamental idea of free and open source software is that code can be modified, and naturally modifications may be used also to fix legal "bugs". In this section we present some options how businesses can deal with packages that are not fully compliant. This is an element that is included in Validos reports, since this is useful information for sharing.

6.1. Removing Problematic Files

Removing problematic files, folders or components from the FOSS package may sometimes be the most efficient method of removing specific legal risks from FOSS packages, caused by, *e.g.*, code which is licensed under incompatible licenses. However, practicality of removing parts needs to be resolved by technical personnel, as incompliant code may be essential to needed functionality or removing code might cause other undesired results such as need for extensive testing. The legal conclusion we have presented above (4.6) discusses legal questions around this.

6.2. Replacing problematic files

Replacing problematic files, folders or components of the FOSS package is closely related to removal of files. Occasionally it can be possible to replace incompliant parts with either compliant versions of needed code or developing such code in-house. Again, practicality of the approach must be evaluated *in casu* since it is dependent on availability of alternative replacements and or amount and costs of developing new code in-house.

6.3. Obtaining Another License

If removal or replacing is not possible for some reason, one alternative which may sometimes resolve incompliance is obtaining an alternative license (FOSS or otherwise) for a code which may not be otherwise redistributable. This option may typically be practical in situations where a FOSS package contains proprietary type software or there is a concern regarding linking copyleft code with other software.

6.4. Accepting related risks

Quite often the legal situation of some FOSS package is subject to true uncertainty caused by ambiguous license terms and lack of relevant case law. These are cases where different but well-founded legal interpretations can be presented but still certainty cannot be reached. Typical issue of this kind is combining and distributing code licensed under the GNU General Public License, version 2 with software that is licensed under different terms. During the years countless number of bytes has been twisted over the issue, but as to date no definite conclusion has been reached.[22] If the situation is subject to this kind of uncertainty, companies can – and very often will – decide on an internal policy and therefore accept related risks.

6.5. Contacting Open Source Projects

Many times simple contacts to the project can solve risk questions. In working with Validos, we have found most projects responsive and delighted of the contribution regarding licensing questions. Sometimes contacting the authors can be a simple way to solve an uncertainty. In relation to our work at Validos, we have not discussed with projects on their willingness to change clear licensing into another type of licensing either for a single case or more generally, but in some cases that could also be an option to consider.

6.6. Refrain from Redistribution

Sometimes none of the above options, or no other measures, are possible or desired. If non-compliance cannot be solved, then the only available option is to refrain from redistributing certain piece of software.

7. Conclusions

Traditional legal analysis, when applied to copyright law in multiple jurisdictions, FOSS environment and package review would find many uncertainties and arguments pro and contra. In this article we have strived to demonstrate another approach: the approach of creating (legal) community consensus around a given methodology or around a set of legal conclusions and thereby controlling risk and enabling and easing the use of FOSS in a business environment. However, such community consensus is not created by one article, but we hope and envisage that this article could help many in creation of their policies, encourage others to criticize and comment the conclusions presented herein and thereby take a step forward in creation of a consensus by the legal community interested in free and open source software.

About the authors

Martin von Willebrand

22 The GPL linking issue has been analysed in a number of publications, including Determann, Lothar (2006): 'Dangerous Liasons--Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs Under Copyright Law, Commercial Licenses, and the GPL' Berkeley Technology Law Journal, Volume 21, issue 4. http://www.btlj.org/data/articles/21_04_03.pdf Accessed on 4 May 2010. and Välimäki, Mikko (2005): 'GNU General Public License and the Distribution of Derivative Works'. The Journal of Information, Law and Technology (JILT) 2005(1) http://www2.warwick.ac.uk/fac/soc/law2/elj/jilt/2005_1/välimäki/. Accessed on 4 May 2010. It is to be noted that a working group of the European Legal Network within Free Software Foundation Europe is in process of finishing a detailed document which examines legal and technical situations of combining GPL v.2 software with other code.

Martin von Willebrand is a partner at HH Partners, Attorneys-at-law, Ltd based in Helsinki, Finland. He has introduced the idea of collaborative open source compliance and is currently the chairman of Validos ry, a Finnish association founded for the purpose of easing the use of open source software by businesses and other entities. He is a technology lawyer who has also significant expertise on copyright matters, open source, and related litigation. His technology work is recommended by practically all publications rating Finnish lawyers, such as BestLawyers, Chambers Europe and Legal500.

Mikko-Pekka Partanen

Mikko-Pekka Partanen is an associate lawyer at HH Partners, Attorneys-at-law, Ltd based in Helsinki, Finland. His main specialisation is copyright and open source and he has extensive experience in legal compliance work relating to software development.

Italian Constitutional Court gives way to Free Software friendly laws

Carlo Piana [a]

(a) Lawyer, Array, Milan, Italy.

DOI: 10.5033/ifosslr.v2i1.38

Abstract
The Constitutional Court in Italy has ruled[1] on the compliance with the Italian Constitution of a Regional law issued by Piedmont on Free and Open Source Software and Open Standards in public administrations. The decision affirmed that a Regional law on public procurement may give more "weight" to tenders that provide Free Software and – by analogy – implement Open Standards. The Court held that such a provision is compatible with the Constitution and, more specifically, that this form of preference is not contrary to competition law. The ruling therefore overcomes one of the most significant obstacles to the blossoming of similar provisions across Europe: that they do not give preference to a technical solution, but rather to a particular type of transfer of rights.

Keywords
Public procurement, Free Software, Open Standard, competition

Info
This item is part of the **Case Law Reports** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

Across Europe, several policy initiatives to implement rules that favour the adoption of Free Software and Open Standards in competitive tenders to public administration have been proposed or implemented. Many reasons have been posited to support such the favouring of such solutions, not least the evidence that proprietary software – through various mechanisms – is unjustly given preferential treatment in many tenders.[2]

1 Decision no. 122 of 22/03/2010 http://www.cortecostituzionale.it/giurisprudenza/pronunce/scheda_ultimo_deposito.asp?comando=let&sez=ultimodep&nodec=122&annodec=2010&trmd=&trmm=

2 As discussed in Rishab Gosh et al, *Guidelines on Public Procurement of Open Source Software*,

Italy is no exception. The main national law that rules on software procurement of the Public Administration[3] is agnostic, and does not go farther than to say that a Public Administration shall always choose between various options – *one* of which is procuring "open source" software – and that the choice should be made according to a technical and commercial comparison.[4] In the national law one cannot find guidance as to how to evaluate the characteristics of the competing offers. This means that any public administration can decide by following the general principles of public procurement.

The Piedmont law was intended to take advantage of the limited but decisive role regional laws have in skewing the situation one way or the other. However, the national government objected to this approach, and the Constitutional Court found that it is constitutionally permissible for a regional law to try to alter the rules of the game of public procurement in order to favour one type of software offer over another, provided that certain conditions are met.

Regional laws

Italy is a federal state. Article 117 of the Italian Constitution defines the legislative powers of the State and those of the regions. There are matters that are reserved to the State, others that are within the powers of the regions, and others upon which the two have concurrent powers. The latter arrangement - under which the present case comes - often causes complex litigation.

Regional laws cannot rule on general private law or on competition rules: these matters are reserved to the State. Regional laws are also not binding to the smaller territorial entities (mainly provinces and communities) within the region, which are entirely independent. But regions have the power to legislate on their own internal rules and of those entities that depend on them. This includes the power to establish, by law, more detailed rules of procurement within the boundaries of the national law. Some regions have decided to legislate on "software pluralism", "open standards" and even "free and open source software" in software procurement. Piedmont is one of them, with possibly the most far reaching provisions. A brief summary of the relevant provisions included in the Piedmont law follows.

The Piedmont law provides that the Region uses software applications of which the source code is available to it and which it can freely modify to adapt them to its needs.[5] In addition – except for the software already in use, in the procurement of software the Region shall give preferential treatment to Free Software[6] as well as to software whose source code is accessible.[7] Finally, if the Region decides to use proprietary software, it shall justify the reasons for such a choice.[8]

http://www.osor.eu/idabc-studies/OSS-procurement-guideline%20-final.pdf Section B.1 (pag. 46, see locally for more references).

3 The "Digital Public Administration Code" ("*Codice dell'Amministrazione Digitale*"), Dlgs no. 82/ 2005, Art. 68

4 So clearly stated by TAR (Regional administrative court) Lazio, Decision no. 428 of 23/01/2007 http://www.giustizia-amministrativa.it/DocumentiGA/Roma/Sezione%203B/2006/200603838/Provvedimenti/RM_200700428_SE.DOC Assoli v. Ministero del Lavoro

5 Art. 6.1

6 The Law uses the words "*Software Libero*"

7 Art. 6.2

8 Art. 6.4

The Piedmont law also favours the use of "open source" software in some of its data protection provisions.[9] Moreover, it provides that for publicly accessible documents, the Region should use "open source" software and "open formats", with a duty to explain any choice to select "closed formats". In such cases, it must also provide an open format version of the document, which shall be as faithful as possible a copy of the closed format document.[10]

All these provisions were challenged by the National Government before the Constitutional Court.[11]

The constitutional challenge

Normally, constitutional cases are brought when the application of an allegedly unconstitutional law becomes relevant in a judicial case ("referral procedures"). However, because the Government represents the central state, it has the right to appeal to the Constitutional Court[12] by raising a "**conflict of powers**" issue. And so it did in this case.

The grounds for challenge between central state and regions are very frequently the fact that, by enacting laws that are in theory within the jurisdiction of National Government, the regions either fell outside the boundaries of their powers in certain provisions, or because by certain other provisions they indirectly violated principles that are fundamental in national law. Both issues were raised in the present case.

The arguments of the National Government concentrated on several specific provisions of the Piedmont law, but some also raised issues with the general approach of the law. Two grounds for annulment were alleged, based on the fact that certain provisions of the law directly fell outside the legislative power of the region, because the provisions regulated certain aspects of copyright law. Copyright law is clearly something reserved to the central state and therefore the law seemed to be doomed. The Court agreed and these provisions were struck out.

Different treatment was reserved to what interests us most here: the favouring of FOSS in public procurement.

The central issue: can Free and Open Source Software be favoured?

The main issue of the case was not on such dubious provisions which clearly ought to be have

9 Art. 5 The law actually uses the Italian wording "*sorgente aperto*", which is a literal translation of "open source".

10 Art. 4

11 The relevant part of the decision is section 6, which deals with the constitutional challenge of most of these three articles.

12 The Constitutional Court is made of 15 judges who stay in charge for 9 years. 5 are nominated by the highest courts, 5 are nominated by the President of the Republic and 5 by the Parliament with a majority of 2/3 of members of the two branches. Ordinary judges, even the Supreme Court, has no power not to apply an unconstitutional law, but they must refer the case to Constitutional Court in case they find than a constitutional issue is not manifestly irrelevant. The Court has over the years (since 1953 when its powers were defined by the first constitutional law) undertook a very important role in the legislative process, sometimes filling in gaps in the legislation and sometimes even creating "new" law by extending the existing ones to unforeseeable fields.

been drafted more carefully: these provisions arguably fell outside the core objectives of the law. The real question was “can a regional government say 'We are going to value any tender that complies fully with open standards and provides the freedoms of Free Software more highly than those that do not'?”

The answer was “yes.”

On this point, the Government alleged that the contested provisions conflict with the national laws on competition. The rules of procurement directly touch on competition issues: this is why the European Union has power to legislate on public procurement, and why many cases are pending against national laws that allegedly conflict with the EU Treaty. Scaling down to the Italian internal market, a similar situation is found. By fixing certain criteria that alter competition in the market, the Region could in theory risk breaching national rules.

The argument of the National Government was that the regional law was against the rules of competition as laid down by the jurisprudence of the European Court of Justice and implemented by the Code of Public Contracts. It can be inferred from the proceedings that the National Government believed that the Region must remain neutral vis-a-vis the different technologies that can compete for a procurement tender.[13] Naming a certain technology against all others is clearly prohibited, and by extension, giving preferential treatment to certain technologies (including or “on the basis of” their licensing regime) should also be prohibited .

The Court disagreed. Here is how it very clearly argues:

> *“The choice is not an exclusive one, but just preferential and requires a comparative evaluation, as is confirmed by the reference to the possibility to use proprietary formats […] under the condition that in such case the Region shall provide motives of its choice [...].*
>
> *Finally, it must be once more reminded that the concepts of free software*[14] *and software with inspectable code are not notions concerning a given technology, brand or product, instead they express a legal characteristic. At the end of the day, what discriminates between free and proprietary software is the different legal arrangement of interest (licence) upon which the right of using the program is based; and the choice concerning the adoption of one or the other contractual regime belongs to the will of the user.*
>
> *It follows that the damage to competition feared by the counsel of the State with regard to the law in question, is not envisaged.”*

13 The actual pleadings are not available at the time of writing, but inferences can be drawn from the motives to reject them.

14 The Court here uses the exact Italian translation of the expression “Free Software”, that is “*software libero*”, and not “*gratis*” or “*gratuito*” (free as in “free beer”).

Conclusion

This is one of the few decisions that tackle directly the possibility of issuing regulations that establish more favourable conditions for FOSS and open standards-abiding software. Proprietary software vendors tend to claim that these rules unfairly discriminate against hypothetically more viable technology and introduce unlawful bias. The Italian Court conversely applies a very reasonable rationale: the authorities, and a regional law, may indeed establish rules to assess not only the technical and economical merit of the offers, but also the **legal rights** that are conferred, and provide greater value to FOSS type licensing. It is a straightforward proposition that can appreciated only when software is considered more than "a product". The licensing conditions of a software product are – as the Court said – a non technical characteristic, not unlike the price or the level of support offered. Nothing prevents a proprietary vendor from choosing a more liberal license or to confer more rights if so weighs favourably. If this is prevented by the upstream licensing conditions, the case is identical for Free Software developers, who also are constrained by the requirement of the upstream suppliers and again it is a matter of choice.

This is the first time in Italy when a regional law that clearly favours Free or Open Source Software is under the scrutiny of a court, and a quite influential one. The decision has been read in many different ways, because it declares certain provisions unconstitutional and certain other valid. But in my reading, the most subtle argument – that based on competition – is the most relevant one, because impacts where the regional laws have more chances to influence the games of procurement. The fact that the arguments have been so clearly rejected by the Constitutional Courts is likely to influence the interpretation of lower courts, including the administrative ones that will decide on the implementation of the general rules laid down by the regional laws. The same reasoning as that of the Italian Constitutional Court seems to be applicable in other parts of Europe where similar policy decisions to favour Free Software exist and where similar objections on non-neutrality grounds are raised.

About the author

Carlo Piana *is an Italian IT lawyer based in Milan as well as a Free Software and digital liberties advocate. Since 2004 he provides consulting to the FSFE and assists the same in battles for competition and open standards. He has represented FSFE and the Samba Team in the antitrust European litigation for obtaining the full interoperability information of the Windows networking interfaces. He is a member of the Editorial Committee of this Review and a strong believer in Free Software and Digital Human Rights.*

Butterworths e-commerce and IT Law Handbook, 5th Edition

Daniel Hopkin[a]

(a) Moorcrofts LLP

DOI: 10.5033/ifosslr.v2i1.33

Keywords
Law; information technology; Free and Open Source Software

Info
This item is part of the **Book Reviews** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement.

Retailing at £125 and consisting entirely of legislation which is freely available online - for example at the UK OPSI's Statute Law Database[1] and EUR-LEX[2] - some might say this book is a waste of trees (whether it's printed on paper from sustainable sources isn't immediately clear). It has its uses, though. For example, there is currently no free way of obtaining access to consolidated versions of UK statutory instruments (SIs, a form of subordinate legislation). The OPSI Statute Law Database updates Acts of Parliament (eventually) but not SIs at present, which means that if you want to look at, say, the Consumer Protection (Distance Selling) Regulations 2000 - a key piece of legislation for anyone selling goods or service online - you also have to look at the Consumer Protection (Distance Selling) (Amendment) Regulations 2005, as well as two other SIs - or pay the subscription for an online service such as Westlaw or LexisNexis. The alternative is to buy this book. In 2,000 pages it covers about 170 separate pieces of legislation, plus things like the domain name dispute policies for .eu, ICANN and Nominet. It doesn't contain common free software licences, though. Adding a few of those would make this an even more useful book for the IT lawyer.

1 http://www.statutelaw.gov.uk/
2 http://eur-lex.europa.eu/

About the authors

Daniel Hopkin *is a solicitor specialising in commercial and IT law at Moorcrofts LLP.*

Moral Panics and the Copyright Wars

Andrew Katz

Partner, Moorcrofts LLP

DOI: 10.5033/ifosslr.v2i1.36

Abstract
Copyright holds a fascination for members of the free and open source community. The copyleft nature of the GPL can exist only because copyright laws exist.[1] This review considers William Patry's book "Moral Panics and the Copyright Wars", and examines its argument that copyright laws are not fit for purpose.

Keywords
Book Review, Moral Panics, Copyright Wars, United States Constitution, Lobbying.

Info
This item is part of the **Book Reviews** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement.

'A "worthless book", "loathsome" and "among the worst books yet written about copyrights"': so said Thomas Sydnor[2] in his rant about William Patry's latest book. My view: Patry's book is beautifully written. His style is accessible and enjoyable. It is impeccably researched: it is well annotated.[3] It is generally restrained (although he does allow himself the occasional outburst from

1 Richard Stallman has found himself in favour of arguing against the Swedish Pirate Party's proposals to radically reduce the copyright term across the board, by instituting an increased term of copyright for free software, without which the GPL's ability to enforce software freedom would be severely curtailed.

2 Sydnor's organisation is called the "Progress and Freedom Foundation": a name which I hope I can assume sounds less ludicrous to American ears than to my British ones. Some of its supporters can be found here: http://www.pff.org/about/supporters.html and include Time Warner, Sony BMG and News International.

3 Almost inevitably, the book contains a few references using Wikipedia links. They are used appropriately (i.e. illustratively, or as a suggestion for further reading, rather than as definitive sources). I'm not at all sympathetic to the argument that wikipedia should never be quoted on the grounds that it is user-generated content and therefore unreliable. However, care needs to be taken when referring to Wikipedia, and I think that links should always be to the version of the Wiki page on the day that it was referenced, or at least there should be a comment in the note, such as www.wikipedia.org/wiki/Aaron_Copland as accessed at 12.01 UTC on 1st April 2009, for example. This minor criticism does feed across to the use of other links: they often have the feel of having been cut and pasted from the address bar of the browser, and therefore may contain the reference variables generating the page in question, rather than a permalink, if one exists, to the page itself. This is more of an observation on the problems of creating persistent links to what may be an impersistent medium, than an indication of any fundamental problems with Patry's work.

time to time).[4] I found no error sufficiently serious to make me question its general accuracy (although there were one or two minor issues which I would either classify as typos, or, if this were a newspaper article, should have been picked up by sub-editors).[5]

Why does an accessible yet scholarly, carefully argued yet playful work generate such vitriol in a reviewer?

Patry's central thesis is simple. Copyright exists for a purpose. We must review the laws that create it from time to time to ensure it continues to fulfil that purpose. Any changes to copyright law must only be made with a view to that purpose being fulfilled. If the purpose is *not* being fulfilled, copyright should be changed, with a view to it continuing to fulfil that purpose. In short copyright should be effective.

Maybe this is such a simple idea,[6] and one for which I can see no counter-argument, that Sydnor feels that the idea is too obvious. So it is curious that at no point during his review does Sydnor challenge the underlying thesis of the book: that copyright should be effective. Patry demonstrates that copyright is not effective, and by examining how we have reached this state of affairs, leads us into some fairly dark conclusions about the failure of the legislative process. It is maybe no coincidence that another prominent copyright expert, Lawrence Lessig, announced that he was changing his direction from copyfighting to changing congress itself.[7]

The US is a fascinating place to undertake an analysis of the effectiveness of copyright, not only because it's a huge market with no shortage of statistics, but because, in the U.S. Constitution, the wise framers explicitly set out the aim of copyright:

> *to promote the Progress of Science and useful Arts, by securing for limited Times to Authors ... … the exclusive Right to their ... Writings*[8]

The European tradition is less fixed,[9] and, in the UK, the lack of a written constitution makes it easier for competing arguments as to the purpose of copyright to be aired. Iain Mitchell QC's fascinating article in the previous edition of IFOSSLR discussing the old case of *Hinton -v-*

4 Pages xxiv and 99 have Patry comparing copyright laws to the sub-prime crisis (and the US Government's reaction to it): maybe not entirely convincingly, but written in an amusingly empurpled fashion.

5 As any Jim White fan knows, David Byrne's record label is called "Luaka Bop", and I'd say that Trent Reznor is Nine-Inch-Nails, as opposed to being their lead singer. Slightly more serious is his contention that Grand Theft Auto sold over $1Bn worth of units in 7 months, a figure that seems excessive.

6 This is a dangerous view to take: Bertrand Russell and Alfred North Whitehead famously took over 300 pages to determine, in Principia Mathematica that 1+1=2 .

7 Lawrence Lessig was counsel for Eric Eldred, the petitioner in the supreme court case of Eldred-v-Ashcroft (http://caselaw.lp.findlaw.com/scripts/getcase.pl?court=US&vol=000&invol=01-618). The case challenged the constitutionality of the Sonny Bono Copyright Term Extension Act in the US. Lessig announced in 2008 that he was quitting the world of copyright, and had decided to aim his sights on an altogether more significant problem: the corrosive effect on democracy of lobbying, triggered, no doubt, by his losing Eldred.

8 The full text of the relevant part of Section 8 is: "To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries". For ease of reading, I elided the references to inventions, which form the basis of patent protection, as that is irrelevant to the book, and this review.

9 Patry doesn't cover moral rights significantly, which is unsurprising as they are more of a European construct than a US one.

Donaldson provides an excellent illustration on this point.[10] But in the US, with this purpose encapsulated in the constitution, Congress is not empowered to pass copyright legislation which does not promote that purpose. Patry, as a U.S.-based lawyer[11] is given a perfect platform to consider whether the system is working as it should: whether legislators are ensuring that the aims of the legislation are in concord with the aims set out in the Constitution.

Copyright exists to promote the progress of science and useful arts: but note, it does *not* exist for the benefit of the authors. Benefit to the authors is the means of promoting the progress of science and useful arts. It's not the end in itself. This is quite a subtle idea: the Framers are proposing allowing a little bit of evil (the granting of a limited monopoly)[12] in order to promote a different public good: the progress of science and useful arts. Like medicine, the dosage is critical.

There are other ways of promoting science and useful arts. Direct state sponsorship or relying on the patronage of wealthy individuals are two of them. But the Framers chose the grant of a limited monopoly.[13] Why they did so is another, and interesting question, but, leaving that to one side for the moment, in the US, copyright exists to promote that clear purpose. It follows that US copyright law can be tested to see if it fulfils that purpose.

And the conclusion if Patry's book is that it doesn't. To be clear, this book identifies the problem, and although Patry's view is that there's too much copyright law (i.e. the monopoly it grants is too extensive) he doesn't seek, in this book at least, to try to quantify how much less is required to achieve the aims (and never professes to do so, although he does comment on a couple of legislative provisions such as renewal requirement under the 1909 Copyright Act). What is clear is that there is no direct correlation between allowing the authors and artists more rights, and the amount of creation that is going on. In fact, I would tend to put it another way. As many authors and musicians I have spoken to have said, their inner muse compels them to write. The economic problem that they have is better expressed as a lack of disincentive (i.e. “I would be writing music, but I've got to go out and get a job flipping burgers at McDonalds otherwise I'd starve”, rather than “if I earned twice as much as I do now from my music, then I'd generate twice as much music”, or “my music would be twice as good”).[14]

So if copyright is excessive, how has it reached that state? If the purpose of the Framers was to progress science and useful arts, how have we come to a situation where copyright laws have been enacted which are clearly much more extensive than is necessary to do that?

10 Mitchell QC, Iain G (2009) 'BACK TO THE FUTURE: Hinton v Donaldson, Wood and Meurose (Court of Session, Scotland, 28th July, 1773)', IFOSS L. Rev., 1(2), 111 – 122 DOI: 10.5033/ifosslr.v1i2.23

11 And Senior Copyright Counsel for Google, Inc. to boot (although I must make it clear that he is writing this book in his personal capacity, and he asks us not to ascribe his views to his employer).

12 The use of the term “monopoly” here is not intended to be inflammatory. Neither is the assumption that a monopoly is evil. These issues were well understood around the time of the framing of the US Constitution, by the Founding Fathers, but particularly Thomas Jefferson. Note also Lord Kames's opinion of the Court of Session in *Hinton -v- Donaldson*, in 1773, three years prior to the Declaration of Independence and 14 years before the ratification of the Constitution.

13 So the constitutional question, and the one explored in *Eldred -v- Ashcroft*, is whether Congress has the power to make legislation, for which has been given certain powers for a purpose, but if the exercise of those powers will not advance that purpose.

14 *A Confederacy of Dunces* was clearly written without the need for any economic incentive, whereas I suspect that *The da Vinci Code* was. I know which I think is the better book. It's fun to play this game, but I admit it doesn't necessarily stand up to scrutiny. Mark Twain, one suspects, wrote to get paid, and so, quite possibly, did Dickens.

In *Eldred v. Ashcroft,*[15] the dissenting Justice Breyer asked in the wonderfully quotable phrase what incentive an additional 20 years of copyright, to start 50 years after the author's death, would generate:

> "What monetarily motivated Melville would not realize that he could do better for his grandchildren by putting a few dollars into an interest-bearing bank account?".

That comment related to a living author; the point was also made that a dead author, whose estate would also benefit from term extension, would find it tricky to dictate a new work from beyond the grave, however incentivised. A number of prominent economists jointly submitted an amicus brief in *Eldred v. Ashcroft* which claimed (amongst other things) that this economic truism was so ludicrously obvious that Milton Friedman initially refused to be associated with it unless it contained the phrase "no-brainer".[16] Again: how could an author who was already dead be incentivised to continue to create more by being given an additional 20 years of copyright protection?[17]

Nonetheless, despite the efforts of Lawrence Lessig on his behalf, Eldred failed (by 7-2), and the copyright term in the US was duly extended.

It's difficult see imagine a more dismal and clear failure of evidence-based legislation.[18] If the legislation envisaged is, in certain significant cases, incapable, on its most generous interpretation, of fulfilling its purpose, then why was it passed? This legislation had received the scrutiny of some

15 See note 7

16 Unfortunately, his wishes couldn't be accommodated: http://www.lessig.org/blog/2006/11/only_if_the_word_nobrainer_app.html

17 There is, I confess, a problem niggling me here. How do we measure the progress of science and the useful arts? This reminds me of the Laffer-curve debates popular during the Reagan era. It's obvious that if the income tax rate is 99.9%, no-one will be financially incentivised to do any work (and will put a lot of effort into avoiding tax, by employing complex tax mitigation schemes, bartering, or simply evading), so tax revenues will be close to zero. Likewise, if the income tax rate is 0.01%, the percentage of tax raised by government will be so small as to be useless, so somewhere between the two there is a magic figure which is the "right" level of income tax, which will generate a maximum amount of tax. It's very easy to use Laffer-esque arguments to say that if there is a certain amount of monopolistic benefit to copyright holders x, then if we increase the amount of monopoly, they will be that much more incentivised. This dramatically over-simplifies a complex multi-dimensional problem, which is probably, by analogy, best illustrated by Martin Gardner's neo-laffer curve: http://en.wikipedia.org/wiki/File:Neo-Laffer-Curve.svg

Finding the appropriate term is by no means easy, and the optimal incentive may vary with the type of work anyway. Another issue is that we tend to assume, copyright in newly created works being, for all practical purposes, infinite, that it retains the same value throughout its economic life, and we tend not to think of what happens towards its expiry.

Lord Kames in *Hinton v Donaldson* was very aware, even in 1773, of this problem. He said, in effect, that even monopolists, where the monopoly is limited in time, would be competing with the future public domain into which their work would fall. If a work is about to come into the public domain imminently, then people will tend to wait until it does come into the public domain, so that they can get it more cheaply. So if you want to sell a book in the last few months, while it is still in copyright, you will have to lower the price, because as copyright expiry gets closer and closer, the lower the premium that people will be prepared to pay to get their hands on the book now. He argued that if a book was never going to make it into the public domain, then this would "unavoidably raise the price of good books beyond the price of ordinary readers".

The Economists' Brief, and most of Patry's arguments, are based on an economic analysis of the benefit. It deserves further thought, but is there another, better, metric which could be used to determine the optimal level of promotion of science and useful arts?

18 Outside the area of drugs policy. Or penal policy.

of the finest legal minds in the US, had been through a lengthy process, including challenge in the Supreme Court, and had still been passed. Is this not a monumental failure of the legislative system? A despairing Patry is, therefore, asking why does this happen? How can it happen?

The answer is that for certain parties, namely the copyright rights-holders (who are sometimes, but by no means usually, the authors), extension of the scope of copyright advances their interests, even if that advancement is at the cost of the rest of society, and does nothing to advance the progress of science and useful arts.[19] Those entities are demonstrably extremely good at persuading legislators to legislate in their interests, and against the interest of society as a whole.[20]

Patry's book, is, in one sense, only peripherally about copyright. What he explains so lucidly and compellingly is that copyright gives us the clearest possible example of a case where legislators are legislating directly against the public interest, in cases where there is no argument of any substance to support their view. In doing so, he carefully considers the toolkit which the lobbyists use.

A significant part of the book deals with the power of metaphor. Human beings are wonderful pattern recognition machines. One consequence of this is that we are good at reasoning by analogy, but just as we are easily capable of seeing patterns where none exists, we are equally good at taking analogy too far. One of the favourite metaphors of those seeking to extend the reach of copyright is the "fruit" metaphor, frequently recast in similarly agrarian style as reaping what one has sown. A recent example of this is the Irish case of *EMI v Eircom.*[21] In which Charleton J repeatedly talks about "fruits", and bases his judgement accordingly. Unfortunately, the Irish constitution is silent as to copyright (including as to its purposes), so we cannot apply *Eldred v Ashcroft* by analogy.

Metaphors are a shortcut to thinking: that is not necessarily a bad thing. The twin constraints of time and cranial capacity mean that no ordinary person is capable of deriving any moderately complex concept from first principles (including, presumably, legislators), and apt metaphors are a useful way of achieving this. However, extending the metaphor too far is fraught with danger. Patry effectively explains the persuasive power of some of these metaphors, and carefully explains

19 Note that incumbents generally welcome regulation of any sort, as it tends to reinforce their oligopoly (incumbents are normally consulted on regulation anyway, but even so, they are best placed to be able to set up the practices and procedures to enable them to deal with it. The costs of complying will be fairly similar for each party anyway, so it will not affect the competitive dynamic, such as it is, between the members of the oligopoly). However, regulation does create a barrier for entrants, who are much greater threat to incumbents, as they are much more likely to behave disruptively,. Note also, that an oligopoly will tend to generate unjustifiably large revenues. Dismantling the oligopoly (even if good for consumers) may be seen as bad by the government department which is regulating them (headlines saying "Ofcom regulation causes mobile telephone revenue to decrease by 20%" are, illogically, not published as good news), so oigopolies are very powerful. Finally, the super-revenue which oligopolies generate may make some things possible that otherwise would not have been. For example it may not have been possible to make the hugely expensive* film "Avatar" in the absence of a Hollywood oligopoly. However, this is not an argument in favour of oligopolies. To quote Eben Moglen: "Without hydraulic despotism and the divine right kingship of the pharaoh, we will underproduce pyramids. Now, we've been underproducing pyramids for three thousand years, and pyramids are beautiful but it isn't hurting us."

* At least the studio saw the benefits of free software. Avatar was rendered on a 35,000 core farm running Ubuntu (http://blog.dustinkirkland.com/2010/01/39000-core-ubuntu-cluster-renders.html)

20 I suspect that Sydor may secretly rate Patry's book as it provides him with a handy "how to" guide to subvert legislators, much as the UK government seems to regard Orwell's Nineteen-Eighty Four as a useful instruction manual.

21 [2010] IEHC 108 Charleton J

why the “fruits” metaphor is inaccurate and misleading.

The book also deals with the persuasive idea that copyright is some sort of natural right: a property right, by which creators should be entitled to be in sole control of their creators. This is a persistent theme in the book. In theory, in American law, it should be irrelevant, unless the proponent is arguing for a constitutional amendment. The constitution makes it clear that copyright is not a natural property right, and therefore legislation short of a constitutional amendments should not try to make it so.[22]

The third weapon which, according to Patry, pro-copyright lobbyists employ is the “moral panic” of the title: examples include John Philip Sousa in the late 19th century and early 20th centuries,[23] Jack Valenti in the 20th century[24] and Thomas Sydnor in the 21st.[25] A moral panic is an unjustified extrapolation from current circumstances to inevitable disaster, and is a well-worn tool of rhetoricians.

Copyright is a technical issue which a relatively small number of people (the readers of IFOSSLR not among them) understandably fail to get excited about. If the rights-holders are able to generate sufficient moral panic to ensure the passage of legislation which will fail to fulfil its intended purpose,[26] then what does this say about the passage of other legislation, where the issues generate more perceived concern in the electorate, such as the Obama's healthcare reforms, or, of more international interest, drug legislation in general? As lawyers as well as concerned citizens, we need to understand how the machinery of legislation and governance can go so wrong.

Patry's book, therefore, advises us to be cynical. Well-funded vested interests are capable of

22 In a sense, the idea that copyright is property is also a metaphor which has gone too far. The Founding Fathers sensibly avoided this analogy in the US Constitution, but in contrast, the UK Copyright Act specifically refers to copyright as a property right (section 1), and while this usefully imbues copyright with some characteristics of property, such as the ability for it to be assigned to bequeathed, it also strengthens the arguments of those who believe that as a property right, copyright is a natural right, and that any encroachment on that right is a form of unjustified government interference from a free market perspective. Patry persuasively argues that, in direct contrast, copyright is nothing but government interference and therefore, and equally from a free market perspective it is the existence and extent of copyright that need to be justified, not any encroachment or limitation on in. This is surely the strongest possible example of the power of metaphor: by a simple recharacterisation from “property” to “government programme”, copyright can be turned from something which free-marketers passionately defend, to something which the same people would condemn.

23 “These talking machines are going to ruin the artistic development of music in this country. When I was a boy...in front of every house in the summer evenings, you would find young people together singing the songs of the day or old songs. Today you hear these infernal machines going night and day. We will not have a vocal cord left. The vocal cord will be eliminated by a process of evolution, as was the tail of man when he came from the ape”. John Philip Sousa in a submission to a congressional hearing in 1906, arguing against mechanical playback devices such as player pianos and gramophones.

24 “the VCR is to the American film producer and the American public as the Boston strangler is to the woman home alone”: Jack Valenti, for the MPAA, giving testimony to the House of Representatives in 1982.

25 The title of one of Sydnor's recent papers is sufficient: “Inadvertent File Sharing over Peer-to-Peer Networks: How It Endangers Citizens and Jeopardizes National Security” (29 July 2009). Available at SSRN: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1443289

26 The Digital Economy Act 2010 has just been passed in the UK, which seeks amongst other things, to grant draconian and disproportionate powers on rights-holders to disconnect copyright infringers without due process. From a technical perspective, it's trivially easy to evade the detection measures that are put in place, and the passage of the Act provides a classic example of the effectiveness of well orchestrated lobbying, and the craven ability of parliament to reject democratic process and subject the bill to proper scrutiny. The lobbyists employed every weapon in the armoury delineated by Patry.

subverting the legislative process. Patry's book tells us how.

One issue which I would like to have seen examined more closely is the fact that income from copyright royalties is easily measurable: losses arising from a decrease in the scope of the public domain are not. Royalties are taxable. Thus an increase in the extent and scope of copyright may well result in a measurable increase in government revenue attributable to one measure.[27] Even though a decrease in the scope of copyright benefits the economy as a whole, that benefit is much more difficult to quantify than the specific decrease in tax take on recording industry profits, for example, and therefore are less likely to spur any individual minister, looking for personal advancement, into action.

I recommend this book to anyone interested in copyright, politics, freedom, democracy, the legislative process, corruption and human nature. It should be required reading for legislators.

Patry has suggested that his next book[28] may suggest how we can fix the problem. I'm looking forward to it eagerly.

27 A related issue is that by enclosing any commons, what was freely available starts generating revenue and becomes taxable. It is also an artificial way of increasing a country's GDP (a statement which, perhaps erroneously, suggests that GDP is not an erroneous concept in the first place). Any economic activity, whether it tends to increase overall wealth of a nation or not, adds to GDP, as Partha Dasgupta recently pointed out (summarised by George Monbiot here: http://www.guardian.co.uk/commentisfree/2010/jan/04/standard-of-living-spending-consumerism)

28 Tentatively entitled “How to Fix Copyright”: http://moralpanicsandthecopyrightwars.blogspot.com/2009/11/my-next-book.html

Lawyers in the Bazaar: Challenges and Opportunities for Open Source Legal Communities

Luis Villa[a]

(a) Legal Fellow, Mozilla Corporation

DOI: 10.5033/ifosslr.v2i1.34

Abstract
There is a fairly common perception among members of the Free and Open Source software (FOSS) developer community that there is no equivalent community of FOSS lawyers. Despite that perception, the FOSS legal community exists and is growing. This note will explore the gap between the perception and the reality, first by briefly surveying the state of the FOSS legal community, and then by exploring the reasons why there is such a disconnect between the lawyers and the community of our clients. Finally, it will suggest some ways in which this nascent community might invest in improving itself.

Keywords
Law; information technology; Free and Open Source Software; community of practice; social capital

Info
This item is part of the **Platform** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

There is a fairly common perception among members of the Free and Open Source software (FOSS) hacker[1] community that there is no equivalent community of FOSS lawyers. Karsten Wade, a deeply involved member of the Fedora community, had this to say on the subject in March 2010:

1 This note uses hacker in the traditional, positive sense of someone who “delights in having an intimate understanding of the internal workings of a system” (http://catb.org/jargon/html/H/hacker.html); as the note will discuss later the author believes that the formal language of 'developers' and 'engineers' is not conducive to community creation.

"During Richard Fontana's talk at SCALE, I heard a clear need for a community of practice for the FLOSS legal community..."[2]

Wade saw a need for a community that in fact already exists, and is growing. This note will explore the gap between the perception and the reality, first by briefly surveying the state of the FOSS legal community, then by exploring the reasons why there is such a disconnect between the lawyers and the community of our clients, and finally suggesting some ways in which the community might invest in improving itself.

The State of the Community

There are a multitude of formal definitions of community. Etienne Wenger provides the following useful one:

"Communities of practice are groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly."[3]

This definition specifically focuses on what Karsten Wade (in the previous quote) and Wenger (in his research) refer to as 'communities of practice' – communities which are formed around skills, rather than more traditional community focuses like a neighbourhood, religion, or political affiliation. The FOSS developer community is a common example of a modern 'community of practice.' The heart of this author's personal, less formal test for community is 'can the members of a group with shared interests call on each other for help when needed?' In both these formal and informal ways, there is definitely a growing FOSS legal community.

First, over the past decade, the progress of FOSS has created a group of legal experts who have a shared interest in understanding and engaging with communities of FOSS hackers. This is a diverse group with diverse motivations,[4] including partners at high-profile law firms, counsel at FOSS-using companies, individual practitioners with FOSS clients, and other interested groups like the Open Invention Network (OIN) and the Software Freedom Law Center (SFLC). Over time, a variety of events, most notably the eighteen month General Public License revision process,[5] have also started building regular lines of communications among these people. The combination of these two trends – increased concern and increased interaction – have resulted in a true community of practice by Wenger's definition. This community has also passed my personal, informal test – when the author's employer (Mozilla) started a process to review its own license, he

2 Karsten Wade, email to Fedora Legal List, Mar. 10, 2010, available at http://lists.fedoraproject.org/pipermail/legal/2010-March/001179.html

3 Wenger, Etienne 'Communities of Practice: A Brief Introduction', http://www.ewenger.com/theory/, via 'The Open Source Way', http://theopensourceway.org/book/single-page/index.html#The_Open_Source_Way-Communities_of_Practice-What_is_a_Community_of_Practice

4 Diversity of motivation is typical of functional communities; for some discussion of this issue, see Benkler, Yochai (2002) 'Coase's Penguin, or, Linux And The Nature of the Firm', 112 Yale L.J. 369; also available at http://www.yalelawjournal.org/the-yale-law-journal/content-pages/coase's-penguin,-or,-linux-and-the-nature-of-the-firm/

5 For background on the process, see 'GPL3 Process Definition', available at http://gplv3.fsf.org/gpl3-process.pdf

was aware of at least a half-dozen people who he knew would answer his call for help. This informal community was very helpful in laying the early groundwork for the Mozilla Public License (MPL) process, and has continued to be helpful as the process has continued.

While the community clearly exists, it is still small and largely informal, largely lacking the sophisticated infrastructure that hackers have come to associate with mature, functional communities. As a result, the community is largely invisible to hackers. For example, hackers look for mailing lists[6] and mini-conferences as signs of community because regular communication is critical (though insufficient) to forming real community. There are now several mailing lists where FOSS lawyers discuss topics of interest. Two such groups – the older Open Bar[7] and the newer Free Software Foundation Europe's (FSFE) Freedom Task Force European Legal Network[8] – have public web presences, but most of them are effectively private – making it difficult for hackers to recognize that such communication is taking place. Similarly, several organizations, like SFLC, OIN, the Open Source Initiative (OSI), and the Linux Foundation (TLF), have started hosting legal mini-conferences, like OSI's recent event,[9] SFLC's series of events in support of GPL v3, and the annual FSFE European Legal Network Conference. As any hacker who has been part of a large community will tell you, such face-to-face conferences are critical for building strong ties – in fact, for hackers, attending such a conference is often a critical step in turning occasional participants into vigorous community members. The existence of this communications infrastructure is a positive sign for the community.

While the community has made some progress in building infrastructure, and therefore visibility, some other structural features that hackers look for in a functional community are still invisible-either missing or, in some cases, private. Most notably, when hackers look to see if a community is healthy and vibrant, they look to see signs that the community is engaging in an ongoing, tangible project; the community's success in building the project is often seen as a proxy for the health of the community.[10] While several FOSS lawyers (most notably those in OIN) are engaged in ongoing projects that might satisfy this role, the specialist nature of the projects may make them fairly opaque to non-lawyers. These ongoing public projects are almost necessarily rare, since most legal projects are one-time events, and many others are private (like most GNU General Public License (GPL) compliance negotiations, or many of the documents produced by the European Legal Network). Because of this they cannot leave the tangible, public artefacts that code-centric communities recognise as signs of success. In addition, because these interactions tend to take place within specific communities, rather than involving multiple FOSS communities at the same time, they may not have a broad impact on community perception.

Relatedly, another sign of maturity that FOSS hackers look for is collaboration with other projects. This signals that others have found that community to be useful and/or easy to work with. The

6 See, e.g., 'The Open Source Way', Sec. 3.2.1.1 "Set up a mailing list first" available at http://theopensourceway.org/wiki/How_to_loosely_organize_a_community#Set_up_a_mailing_list_first.

7 Open Bar, web site at http://www.open-bar.org/

8 Freedom Task Force, web site at http://www.fsfe.org/projects/ftf/index.en.html

9 Radcliffe, Mark, 'February 10 FOSS Legal Strategy Session Silicon Valley: Success!' available at http://lawandlifesiliconvalley.com/blog/?p=421

10 Traditionally journals like this one serve a similar signaling functions in the legal community by conveying to other lawyers that a sub-specialty has reached a critical, self-sustaining mass and a sense of self-identification. It will be interesting to see if this journal performs a similar signaling function to the hacker community.

most efficient way to do this is to provide something that is useful to many projects at once; for example, the Linux kernel and GNU compiler are viewed as successful and mature in part because the software they provide underpins so many other projects. The FOSS legal community is not without examples of this – the GPL, for example, is used by many projects, and the SFLC's Conservancy[11] is becoming more widely known as a good example of the kinds of useful infrastructure that lawyers can provide to developers. But these examples are rare, and that rarity contributes to the visibility and awareness problem that faces the FOSS legal community.

Legal Culture and the Bazaar

The lack of community infrastructure and recognition described in the previous section is in large part caused by the short time that lawyers have been involved in FOSS. These things don't come overnight, even to the most 'natural' communities. But it is also worth examining the structural and cultural barriers which have helped make it hard for lawyers to participate in the open source 'bazaar.'

Before focusing on lawyers in particular, it is worth reviewing the concept of the bazaar. In Eric Raymond's seminal 'The Cathedral and the Bazaar',[12] he argues that pre-open source development was focused on a model of development that resembled that of the cathedral-builders, where only certain participants were allowed to contribute, and then only in a very structured, hierarchical fashion. In contrast, he observes that most open source projects are more like a bazaar – “a great babbling bazaar of differing agendas and approaches” where anyone who wants to can set up a 'stall' and eagerly exchange knowledge, skills, and content. While this 'bazaar' model is certainly not perfect,[13] it has proven quite robust and is the dominant organisational model for most open source projects. In contrast, legal practice has traditionally been very 'cathedral'-like, and this culture will put some barriers between lawyers and open source clients.

The first example of problematic legal culture is the common practice of the billable hour. The billable hour encourages us to be very sceptical of any work which is “inefficient” or which doesn't give an immediate pay-off. Unfortunately, much of the work necessary to build social capital in a community falls into one or both of these categories. For example, educating others to do common tasks is one of the things that healthy communities of practice do well. This process is costly in the beginning (because work is done by less efficient workers, and the expert's teaching time is time not directly applied to the problem at hand) but has long-term pay-offs as the community of skilled individuals grows, allowing experts to focus more exclusively on high-value tasks.

A second, related issue is the occasional arrogance lawyers have about the skills necessary to do legal work. In the non-FOSS world, this problem manifests itself as a reluctance to use paralegals for many tasks leading to unnecessary client costs. In the FOSS world, there is a slightly less

11 Software Freedom Conservancy, website at http://conservancy.softwarefreedom.org/
12 Available at http://catb.org/~esr/writings/homesteading/cathedral-bazaar/
13 See, e.g., discussion of the implications of the bazaar model for user interface design in Feinberg, Jay, 'The cathedral and the bazaar of user interface design', available at http://icite.net/blog/200403/cathedral_bazaar.html and Dörner, Christian, et al, 'End Users at the Bazaar: Designing Next-Generation Enterprise Resource Planning Systems', available at http://doi.ieeecomputersociety.org/10.1109/MS.2009.127

direct impact. Healthy communities break down the barriers between 'insiders' and 'outsiders'[14] so that new bodies, new perspectives, and new enthusiasm regularly refresh the community. Unfortunately the elitism of the legal community can make this hard to do. This is not an easy problem to solve, of course, since much of what lawyers do is in fact extremely complicated, and in many cases requires not only formal training but also extensive practical experience. Despite this problem, the impact of this insider/outsider barrier can be reduced. For example, a healthy legal community could explicitly seek to provide tasks and other entry points that are accessible to non-lawyers. (Most distributions do this to some extent by farming out very basic licensing testing to hackers.) A healthy community could also clearly articulate why some tasks truly do a specific skill set, making new participants less likely to be discouraged when they aren't able to participate. Taking these steps would help broaden the base of participants, giving the community more resources in the future.

The lawyer's habitual allergy to plain English,[15] which is so essential to a functional community that it isn't even mentioned in The Open Source Way, is also a problem. Non-hierarchical communities tend to form around people who communicate in the informal style which is naturally associated with relaxation and fun (in the case of volunteers) or associated with peers relating to each other on a fair basis (in the case of multi-company collaborations). This communication may be highly jargon-filled (when that jargon is central to the mission of the community) but that should not be confused for formality. Formality is typically a signal of distance, hierarchy, and other things that aren't conducive to community formation. The lawyer's natural tendency to be both formal and use a lot of jargon is a dual killer.

On top of all these other problems, the structures of privilege, evidence, and ethics law combine to make lawyers very nervous about discussing anything in public. Young lawyers are often explicitly told that doing things in public has very few upsides and vast numbers of potential downsides. This means that even when all other cultural barriers are reduced or removed, there is a tendency to keep the communities that do exist – and their successes – private and out of the limelight.

All of these cultural factors combine to reduce the visibility of the FOSS legal community, and reduce the community's ability to interact with and support other open source projects. While no magic bullet can cure these problems, awareness of them, particularly when planning open source legal projects, could help reduce their impact and increase the success of the legal community.

Digression: Should FOSS Lawyers Seek Recognition?

This note has presumed that increased visibility and effectiveness would be a good thing for the FOSS legal community. While increasing the social capital and infrastructure of the legal community is unlikely to be controversial, the other part – increasing the broader FOSS

14 See, e.g., discussion in 'The Open Source Way', http://www.theopensourceway.org/book/The_Open_Source_Way-Communities_of_Practice-Open_a_dialogue_between_inside_and_outside_perspectives.html

15 For the web user's take on the use of plain English, see Levine, Rick, et al, (1999) 'The Cluetrain Manifesto', available at http://cluetrain.com/. For a specialist's review of the harm done to the legal profession by specialized language, see chapter 4 ('Some Benefits of Drafting in Plain English') of Butt and Castle (2006), 'Modern Legal Drafting: A Guide to Using Clearer Language'

community's awareness of the legal community – is clearly not a unanimous goal of the legal community.

Increased legal visibility in the FOSS community could be seen to have two primary downsides. First, increased legal participation could be seen as a source of increased transaction cost between open source participants – more documents to read, more risks highlighted, etc. This is a legitimate concern; after all, lawyers do not have a sterling reputation for reducing transactional friction. This should not be an overriding concern, though. The success of the GPL and Creative Commons suggests that *when lawyers are sensitive to the problems of friction* they can use techniques like standardisation, simplification, and attentiveness to client needs to increase trust and reduce transaction costs.

The second theoretical downside is that increased legal participation could be interpreted by naive consumers as a sign that there is increased legal uncertainty around FOSS. Since increased legal uncertainty is a known strategy of the opponents of open source,[16] this is a legitimate concern, but not an insurmountable one. For example, the community could reduce this risk by turning the spotlight from work on high-risk issues, like patents, to issues that play to FOSS strengths, like inefficiency-reducing standardized documents.

These potential downsides should be weighed against the many values of publicity. Most notably, knowing that lawyers are working actively on their behalf will help increase hacker community confidence – and hence investment in – their own work. Such confidence is an explicit goal of projects like OIN. Publicity will also help bring more resources 'out of the woodwork' – as any experienced community participant will tell you, other people can't help if they don't know what you're working on.[17] These benefits should outweigh the concerns discussed above.

What next?

In hacking, the shift from a cloistered, do-everything-in-house mentality to one focused on public collaboration and sharing did not happen overnight.[18] Making the same shift will be even slower for lawyers because of the structural and cultural barriers this note has highlighted. These barriers are generally deeply engrained; some may not go away at all and others will go away only slowly. But, keeping those concerns in mind, the rest of this note will suggest some projects that might be reasonably attainable while also increasing the long-term effectiveness of the FOSS legal community by broadening and deepening the community's social capital.

16 See, e.g., Parloff, Roger, 'Microsoft takes on the free world', Fortune Magazine, May, 14, 2007, available at http://money.cnn.com/magazines/fortune/fortune_archive/2007/05/28/100033867/index.htm/

17 In a recent example of this, one major open source company recently announced a project to collaboratively build a specific type of legal document which could be reused across many projects, reducing transaction costs for everyone who used it. This announcement immediately won commitments from a substantial number of legal community members to work on the document. Less than two weeks later, another major open source company announced that they had been privately working on a nearly identical document for a year. Had the second company announced their work earlier, the document would likely have drawn resources from outside the company (making the process more efficient for them) and made the resulting document more useful for a broader range of projects (making the outcome more useful for the open source community as a whole.)

18 For a pre-open source discussion of this phenomenon in Boston and San Francisco, see Saxenian, Anna-Lee (1996) 'Regional Advantage: Culture and Competition in Silicon Valley and Route 128', Harvard University Press

The first category of projects would be those that lower the barriers to community formation and participation by lawyers. This category of project is perhaps the one where the most work is already going on, led by various groups who are bringing lawyers together at conferences to educate them and build social ties. But the community could go further. One potential project is in the area of privilege law. As already noted, many lawyers believe that privilege law makes it difficult to give legal advice to an entire FOSS community. No formal guidance (that the author is aware of) lays out how and to what extent lawyers are legally and ethically permitted to give public advice to ill-defined communities; instead, lawyers fall back to their own conservative instincts in this area. If a group of lawyers sought and publicized formal ethical guidance in this area, more lawyers would be enabled to do more work with pre-existing communities.

Another category of projects that could be considered are educational or 'security blanket' projects that seek to protect and calm the broader developer community by educating the community about intellectual property law, combating disinformation, and educating the community about the legal community's protective strategies. For example, at a recent meeting of FOSS lawyers, there was discussion about patent strategy – not merely planning for the next attack, but also about educating and informing the broader community about the plan. Such a proactive, two-pronged approach which explicitly combines action with broad education about the action would help get the legal community itself in the habit of working collaboratively on such projects as well as making the community aware that good work was being done on their behalf. This would make members more competent in self-assessment, less fearful, and more motivated to do their own work.

Collaborative projects that produce reusable legal documents are a third category of work that could help build legal community social capital as well as directly benefiting the developer community. Interestingly, this community has produced some of the most widely used legal documents on earth (in the form of the various public copyright licenses) but has largely failed to produce similar standard documents dealing with common issues like basic legal and intellectual property education, trademark licenses, privacy policies, and copyright assignments. Collaborative attempts to write such documents would not only help reduce friction for the community of FOSS practitioners, it would again build the legal community's experience, enabling it to more effectively deal with future challenges.

Finally, lawyers should look to identify legal work that can engage non-lawyers, both because of the pragmatic short-term effects of engaging a large community with a lot of resources, and because of the long-term effects of building a more impactful community. One example of this might be engaging with the engineers who work at distributions to do a more effective job of checking license compatibility in the projects that they package for the distributions. Doing this would leverage pre-existing engineering resources to improve the quality of licensing information available to the entire community. In the same vein, OIN has a project which attempts to collect ideas from hackers and turn them into records of defensive prior art[19] available to patent examiners. This is an excellent example of the kind of work that can be done by helping non-lawyers create information. One could easily imagine extending or duplicating this model for a variety of legal projects which need engineering input, such as attempts to explain linking to lawyers.

19 Linux Defenders project, website at http://linuxdefenders.org/

This list of potential projects is certainly not exhaustive, but is intended to illustrate the kinds of projects which could lay the groundwork for increased effectiveness and visibility for the FOSS legal community. Careful thinking about cultural and structural ramifications of every project, not only the legal ramifications, can help build a more effective long-term community – of both lawyers and clients.

Conclusion

Despite the barriers identified in this note, the FOSS law community is still growing, which is a testament to the power of the collaborative model and the social and economic incentives which are drawing lawyers to collaborate. Hopefully this note can contribute to this growth by spurring the community to greater introspection when planning projects and considering communication with this atypical, but worthy, group of clients.

Oriented Specification System

Pieter Hintjens,[a]

(a) Digistan

DOI: 10.5033/ifosslr.v2i1.32

Abstract
Pieter Hintjens tells the story of COSS (the Consensus-Oriented Specification System), a toolkit for emerging digital standards. COSS gives workgroups the tools to develop new digital standards with minimum bureaucracy. COSS is a product of the Digital Standard Organization (Digistan), which the FFII open standards workgroup founded with other participants in 2007. In this article he explains why Digistan built COSS and how it works.

Keywords
Law; information technology; Free and Open Source Software

Info
This item is part of the **Platform** section of IFOSS L. Rev. For more information, please consult the relevant section policies statement. This article has been independently peer-reviewed.

Standards are essential to any industry. By acting as contracts for interoperability, standards free customers to choose among suppliers. Standards thus create competition among suppliers that forces those suppliers to improve quality and/or reduce prices. By slowing and stopping disruptive innovation in layers where it is no longer needed, standards allow layering of new industry segments on top of old ones. For example, wiring and voltage standards for electricity underpin the business of selling electrical appliances, and standards for shipping containers massively increase the efficiency of the logistics and transport industries.

Standards-setting processes also reflect the nature of the specific industry concerned. Industrial standards are usually set by consortia, often backed by state regulation. For some industries such as telecommunications, standards are a prerequisite for development, whereas other industries develop standards later, in mature areas. George Stephenson may have opened the Liverpool and Manchester Railway in 1830, but the UK did not converge on a standard gauge until 1892.[1]

1 http://en.wikipedia.org/wiki/Standard_gauge

Software standards, and the processes which facilitate their development, reflect the nature of the software industry. In particular, the conflicts between old and new money, between large and small organizations, and between command-and-control and collaboration are evident.

Free and Open Source Software (FOSS) has been described by some - mainly proprietary software businesses -- as a "business model". That seems inaccurate. Most collaborative projects have no initial business plan beyond "1. Popularity, 2. ???, 3. Profit!" but many succeed nonetheless. It is more useful to view FOSS as a social/legal technology for software development. Every FOSS community depends on a contract that encourages or forces participants big and small to share their knowledge and work. Perhaps as importantly, end-users are brought into the social contract. Giving end-users the ability to inspect, improve, and if necessary, fork the source code means that they can help steer the direction of development. This happens even in small FOSS projects, if end-users get involved early on. Decisions about whether or not to add new features are more accurate when users are involved in the process.

It appears that the FOSS social practices and legal infrastructure – primarily represented by software licenses like the BSD (Berkeley Software Distribution license) and GPL (GNU General Public License) – result in faster technological development than the conventional proprietary software approach. We do not have studies to prove this, only experience and observation. One recent visible example is the Linux-based open source Android operating system for mobile phones, which seems to be overtaking its proprietary rivals both in rate of innovation (in June 2010 we see four versions on the market: 1.6, 2.0, 2.1, 2.2) and in market success[2]. Other open, collaborative projects such as Wikipedia provide more dramatic success stories.

Small FOSS teams often need to compete with larger established firms in the market[3]. This requires a divide-and-conquer approach (many smaller teams competing with a few large ones), which requires teams to agree on shared file formats, protocol, APIs, and languages so that components from different teams can work together in larger systems. These agreements will be more widely used and respected when they are properly written, formally endorsed by some body, accessible to all, and free of all constraints on use.

The supplier of a commercial product facing FOSS competition may arbitrarily change interfaces, file formats, protocols and such, to deny interoperability and thus keep customers captive. The classic example is the format for Microsoft Office documents, which started changing in each release of the software, when open source competitors began to accurately read and write files in that format. When these interfaces, file formats, and protocols are documented and freely usable, customers have a means by which to demand compatibility from the supplier. Without a written specification, the implied contract provides "compatibility with previous versions" at best, and suppliers can always introduce change under the excuse of "innovation." With a written specification, the implied contract becomes "compatibility with the specification."

Thus, properly written specifications of interfaces and formats (even without the backing of a formal standards body and even without cooperation of all vendors) are a key part of allowing

2 "Android overtakes Apple in US smartphone market", http://arstechnica.com/apple/news/2010/05/android-overtakes-apple-in-us-smartphone-market.ars

3 http://en.wikibooks.org/wiki/FOSS_Government_Policy/Strategic_Importance_of_FOSS

FOSS teams to compete in scale with entrenched vendors. Even within a single FOSS project, documented specifications for interfaces, protocols, and formats create contracts between developers who may work far apart in space or time.

Microstandards

The first "Requests for Comments" (RFCs)blished 30 years ago, were small software specifications that solved specific pieces of a vast decade-long puzzle: how to build a global computer network available to all at a cost approaching zero. These RFCs were built by small teams, often just one or two people, using a modest process based on peer review and rapid maturation. They escaped the normal commercial pressure to turn the standard to favour any particular vendor In contrast, the industry consortia that were solving basically the same problem developed heavily patented telecommunications standards that created a captive market, dominated by incumbent telecommunications firms.

There can be little argument that this approach was wholly successful, because every competing networking technology, developed at great cost by major firms like IBM and international standards development organisations, lost against the Internet RFCs. Standards such as token-ring, LU6.2, SNA and VTAM were quickly eclipsed by the open RFC approach. The few areas where proprietary networks still dominate (such as mobile telephony) are remarkable for their high cost and lack of true competition. In 2010 Engadget.com reported[4] on the high cost of international mobile telephony, and especially roaming mobile data, concluding "*Do not ever, under any circumstance, roam with your American mobile broadband card. You'll never pay off the roaming bill.*"

The term "microstandard" means, in this context, a free and open standard that is short (perhaps a dozen pages and certainly less than 100), developed by a small team, and part of a stack of free and open standards. In general microstandards can evolve faster thanks to rapid maturation-layering cycles; they can be more accurate (meaning: good solutions to real problems) thanks to review by a wider, more diverse community; and they can be cheaper to implement and use, thanks to simple, focused designs. However, these qualities do not guarantee success, and only a minority of potential microstandards are ever properly written down, published, shared, and built upon. If we were to count, we would find tens of thousands of potential microstandards worldwide, most of which never emerge from a niche: little scripting languages, data schemas, file formats, ad-hoc protocols, encodings, patterns, templates, and APIs. This informal economy is massive, but it is also massively inefficient. Most of these potential microstandards exist only as source code.

The vast bulk of these microstandards remain potential and local because their authors lack experience and economical tools for properly writing and maintaining standards. Doing so is just too hard and too costly. The inherent friction between innovation and standardisation has turned into a barrier against innovation in the standardisation process itself.

This has been my experience in three decades of software architecture and programming: where

4 http://www.engadget.com/2010/06/09/how-to-stay-connected-while-traveling-internationally/

there could be tens of thousands of properly built microstandards, each clearly documented and published under standard licenses, we see barely dozens or hundreds. The standards process is in my view failing at the grass roots level, exactly where the FOSS economy needs it most of all.

Software standards have been the focus of conflict between smaller FOSS teams and larger established players, and these conflicts are starting to interfere with standards setting at the international level. The key case is the ISO (International Organization for Standardization) adoption, over several years, of two conflicting formats for office documents.

Standards War

In 2007, a world-wide standards war broke out over document formats. On the one side, Microsoft, the largest global software business, was advocating its OOXML (the confusingly named "Office Open XML"). On the other, a coalition of activists, engineers, and FOSS businesses advocated the adoption of the Open Document Format (ODF). In the end, after a year of massive, global conflict in hundreds of national standards committees, and despite a well-organized global campaign by the pro-ODF coalition, OOXML was given the ISO stamp of approval. It was a dirty fight in which committees were stuffed, coerced, and bypassed. ZDNet wrote[5]:

> *"What now transpires is that [Microsoft] have hijacked the committee and are not only stepping outside the established procedures but are also working towards amending the standard in order to make it compatible with Office 7, rather than building or amending their Office Suite to be compliant with the ISO standard. Apparently, it's only through leaks the we can find out what's happening."*

For most of the conflict, the coalition was trying to decode the rules that defined who could vote, and when. At the same time Microsoft was rewriting them to ensure it would win any final vote, in any case. A typical story came from Denmark, which, as ZDNet reported[6] "decided to back Microsoft's Office Open XML document format, reversing its previous disapproval and bringing the format closer to fast-track approval by the International Organization for Standardization." Wikipedia documents[7] complaints about national processes in Norway, Sweden, Portugal, Finland, Switzerland, India, Australia, Germany, Poland, Spain, and the UK. In Poland, says the Wikipedia article, "*Borys Musielak, a member of Poland's Linux community, wrote on the PolishLinux website that Poland's technical committee KT 171 rejected Office Open XML. The vote was invalidated and assigned to KT 182. A member of Poland's Linux community believes this was due to "reorganisation in the Polish standardisation body." KT 182 voted to approve Office Open XML.*" The NoOOXML.org campaign followed and documented these and other similar stories via hundreds of contacts across the world. That documentation is still online and accessible via the NoOOXML.org website.[8]

5 http://www.zdnet.co.uk/blogs/moleys-musings-10008506/ms-stuffs-ooxml-jtc1sc34-maintenance-committee-10014322/
6 http://www.zdnet.com/news/ooxml-standard-vote-down-to-the-wire/194601
7 http://en.wikipedia.org/wiki/Standardization_of_Office_Open_XML#Reactions_to_standardization
8 http://www.noooxml.org

Many people were left wondering if the traditional standards business could still represent the needs of the modern software industry or whether they were to be dominated by narrow but powerful individual corporate interests.

Standards processes appear to reflect their industry as a whole. The great shift over the last twenty years has been from traditional ways of making software to the collaborative approach of FOSS. It has become evident to many in the industry that FOSS is a better way to make software, producing more accurate solutions faster and cheaper than older closed source approaches. The difficult question is rather: "how do we make money when the software is free?"

For monopolists like Microsoft, the answer is to prevent the software from being free at all. This requires a number of mechanisms, including software patents, EM (original equipment manufacturer) licensing agreements, proprietary languages and frameworks, and arbitrary changes to key file formats, interfaces, and protocols. If governments and businesses insist on those being standardised, that requires control of the standardisation process. If governments and businesses insist on ISO standards, that eventually means controlling the relevant parts of ISO itself.

And so in early 2008 the unthinkable happened and Microsoft effectively took over parts of ISO, both directly and through proxies. Most national committees and the ISO secretariat were persuaded, bullied or otherwise encouraged to accept Microsoft's proposal with minimum changes as a formal international standard. The anti-Microsoft coalition was just as ready to use political influence and committee stuffing, but failed to appreciate the depths to which Microsoft was willing to go in order to save its Office monopoly.

Lessons from the Standards WarThe relevance of this case is as a record of how easy it was for a determined large firm with deep pockets to turn what had been an advantage for proponents of ISO certification of ODF into a direct liability. Since Microsoft now in effect controls ta sufficient number of national standards-setting organizations, it effectively controls ISO standards setting, and thus it controls ODF and is positioned to slowly strangle it.

Participants in the process (both in committees and outside) took home various conclusions from this dramatic series of events. Many decided that it was time to make peace with Microsoft. Others decided to create new institutions like the Open Web Foundation. Others, and especially the activist core of the NoOOXML.org campaign, came to the conclusion that ISO and other institutional processes had become irrelevant, if not actually distracting time wasters for standards engineers working on open standards, often *pro-bono*. For example, after years of work by participants around the world to get ISO approval of ODF as the standard for document formats, this very success may have spelt the death of ODF.

The activists, among them myself, realised that any standards-setting institution, including the W3C(World Wide Web Consortium) and IETF (Internet Engineering Task Force), represents a target for determined corporate attacks of the kind we had witnessed during the document format standard-setting process within ISO. Ultimately, any process that depends on goodwill is vulnerable to hostile takeover.

What had been missing, we concluded, was a standards-setting process that matched the fully distributed, attack-resistant processes of the FOSS ecology. It was not enough, we felt, to be free today, we needed the unambiguous guarantee of freedom tomorrow. As Andrew S. Groove said, "Only the paranoid survive."[9]

Thus, in 2007 we created the Digital Standards Organization (Digistan) with a mission to "promote customer choice, vendor competition, and overall growth in the global digital economy through the understanding, development, and adoption of free and open digital standards". Our first publication was a new definition of "free and open standard" based on analysis of the standards development process.[10] The core definition is that "*a free and open standard is a published specification that is immune to vendor capture, at all stages in its life-cycle*".[11] “Vendor capture” is a term chosen to focus attention on the economic interests of those who make standards. Those making money from products – vendors – have an economic interest in selling more products by reducing competition. They can achieve this by taking control of the standard – capturing it – in various ways.The language of Digistan's definitions is heavily influenced by the standards war of 2007, because of the enormity of this event in shaping our understanding of the economics of standards development.

The Effects of Vendor Capture

Frequently, a group of firms will create a standards-setting organization and then exclude upstart competitors from participating. With the addition of some patents, even though the patents may be made available under a so-called “reasonable and non-discriminatory” licensing scheme, this creates a neat and legal cartel that is immune from serious competition authority oversight. Even when there is flagrant and long-term extortion of the market, patent pools make legal behaviour that would otherwise result in prison terms[12]. The mobile phone sector is a case worth studying.

Ewan Sutherland of the LINK Centre, University of Witwatersrand and CRID, University of Namur, writes[13] that in 1996 the European Commission (EC) opened its first dossier on international mobile roaming (IMR), at the behest of the mobile operators, who sought and received an exemption from anti-trust laws of the EC Treaty (now Article 105 (1) EU). In 1999 the International Telecommunications Users Group (INTUG) filed a complaint with the EC about persistently high prices. Cases against four operators were closed without any penalties, and an inquiry into the handling of these cases was abandoned, with no explanation to the high prices. There were a set of directives in 2002, and two more Roaming Regulations in 2007, but prices for IMR remain high, especially for roaming data. Sutherland concludes "*The mobile network operators maintain there was no problem. The EU institutions,... and many users maintain that*

9 http://www.intel.com/pressroom/kits/bios/grove/writings.htm
10 http://www.digistan.org/text:rationale
11 http://www.digistan.org/open-standard:definition
12 http://www.crowell.com/documents/Antitrust-Risk-in-Patent-Pool-and-SSOs-Avoiding-Price-Fixing-and-Exclusionary-Conduct.pdf
13 The European Union Roaming Regulations, http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1574981

there was and is -- even if they can only describe the symptoms, not explain its causes."

To uncover the causes of the problem, and why national and EU regulators have been unable to correct what looks very much like collusion by dominant market players, we need to understand that the mobile telephony market depends on standards, and these are very heavily patented. As Tobias Kaufmann explains[14], the GSM (Global System for Mobile Communications) market depends on 554 patent families (not individual patents) declared “essential” by their holders. 74% of these patents are held by four firms.

Kaufmann explains how patent licensing is used to create the cartels that dominate the European and American mobile phone markets:

> *In 1998, the ITSUG (International Telecommunications Standards, User Group) authored a complaint to the European Commission in which it summarized the GSM licensing problems in stating that the inability to acquire timely licenses coupled with the uncertainty of essentials lead to "costly and complex licensing negotiations" and "excessive cumulative royalty rates", thereby creating substantial transaction costs and high barriers to entry to the GSM market. In addition, the ITSUG alleged the existence of a "low/zero club for established European telecommunications players" while others have to pay royalties of up to 40% of the ex-works selling price".*

So while GSM depends on standards, those standards and the market they depend on are entirely captured through patents, and escape normal competition pressure to lower prices. By adding new patents to the pool of patents that a business must license, the normal 20 year lifespan of a technology like GSM – which originates from 1991 – can be extended almost indefinitely, and indeed Kaufmann reports that only 149 of the 554 patent families in the GSM patent pool are judged essential by technical experts. The rest exist only to perpetuate the licensing barriers. As Wikipedia states[15], “*Patents remain a problem for any open source GSM implementation, because it is not possible for GNU or any other free software distributor to guarantee immunity from all lawsuits by the patent holders against the users. Furthermore new features are being added to the standard all the time which means they have patent protection for a number of years.*”

In a properly competitive market based on free and open technological standards – such as the traditional wired phone network – we would expect costs to fall by 50% every 18-24 months, or value to double every 18-24 months (hence “Moore’s Law”, stating that chip density will double every two years[16]. If it cost 1.00 UKP to make a 1-minute international phone call in 1990, today that cost should have fallen by half, ten times by 1000 times.

Free and Open Standards

The "immunity from capture" analysis is key to Digistan's work. Many definitions of "open standard" work by defining properties of an open standard. This mixes symptom with cause. A

14 Intellectual Property in Broadband Mobile Telecommunications: Predictions on 4G WiMAX , http://www.frlicense.com/IntellectualPropertyinBroadbandMT.pdf

15 http://en.wikipedia.org/wiki/GSM

16 http://en.wikipedia.org/wiki/Moore%27s_law

standard may be open because it is not yet captured. Or it may be open because it cannot be captured. We need to express and measure its immunity from capture, not the consequences of "so far, so good". For example common term “open standards” is often stretched to include standards with “reasonable and non-discriminatory” (RAND) patent licensing conditions.

Standards have a lifecycle that often lasts many years. At various stages of the lifecycle, different tactics can be used to try to capture the standard. For example, a standard developed by a community, free to read and implement, can be trivially captured by a single patent, by a decision of a copyright holder to start charging for updates, or by an injection of complexity that makes it impossibly hard for independent vendors to implement.

We would argue that de-facto, any standard that affects a significant market will be subject to such capture attempts. In Digistan we therefore defined this new term, "Free and Open (Digital) Standard" to express this additional criteria. A free standard will be open, and continue to be open. A standard that is only “open” (now) is not necessarily free, nor open in the future.

The resonance with "free and open source" was not deliberate but it is accurate, and one could argue the same cause-and-effect relationship between "free" and "open" in software.

What makes a free standard? Here we diverge from conventional standardisation wisdom and take an idea that has been wildly successful in FOSS projects, where it plays an essential role in preventing certain forms of capture: the right to fork.

The Right to Fork

We realised that forking was a necessary right when studying the ways that a standard could be captured at different phases in its lifespan. It seemed a counter-intuitive proposal at first, and one that provoked extensive debate among the Digistan founders. Even a small, young standard exists because it is definitive. How can the right to fork make standards better? Surely it would result only in arbitrary, destructive variations?

Ultimately any specification must aim to be definitive and to get approval from a body like the IETF, but such approval may not be achieved for five or more years after the specification is documented and implemented. Our goal is to ensure that a specification survives those early years and gets ready for international certification unharmed and 'un-captured'.

There is a natural tension between experimentation and standardisation. These are often described as opposing processes. In fact, they are just different stages in a lifecycle. Early on, engineers must experiment in order to learn what the best solutions are. As they learn, they capture their growing body of knowledge as standards. Each standard, or stable version of a family of standards, encapsulates some layer of knowledge, freezes it, and creates a solid basis for constructing further layers.

Capturing one of these building blocks effectively captures everything that depends on it, much as buying a software company effectively buys the market for that software.In both cases, the right to fork is leverage that encourages good governance. If a firm that buys a FOSS project makes bad decisions, key developers and contributors fork the project, and walk away.

Here is a practical example of how the right to fork helps keep a standard open: firm A develops a standard in a lucrative new market. Firm B buys firm A and acquires the copyrights of the standard. It then creates a new version that is compatible with firm B's products, and not those of competitors. It uses patents to prevent competitors implementing that modified version of the standard except for a heavy fee. It uses copyright to prevent further modification of the standard to avoid the patented aspects. In effect, firm B has captured the market and everyone who built on the original standard faces the choice of stagnation, or the higher prices charged by firm B.

Now, in a second scenario, the standard was developed under a license guaranteeing the right to fork. When firm B buys firm A with the intention of controlling the market, other implementers fork the standard. They create a new version that bypasses the patents, and it is that modified version that is promoted to the market for adoption. Firm B has no mechanism with which to capture the market, so instead it competes with better products at lower costs or it is forced to conform to the forked standard.

GPL for Free and Open Standards

The idea that forking was a solution rather than a problem in standardisation processes informed our search for an appropriate license for standards texts. We consulted the Free Software Foundation (FSF), who told us that the FSF did not have a license appropriate to standards specifically. In the end we chose the FSF’s General Public License (GPL) v3.[17], to avoid license proliferation.

Using the GPL for a specifications text is unusual but we can make more sense of it if we imagine that a specification might often be a kind of software. For example, a specification may include an XML schema or a set of formally described methods and arguments. Many modern standards start to look like a mix of metadata and comments, and thus start to overlap with software source code.

To apply the GPLv3 to a specification, we state that the specification is "source code" and we ask that contributors license their work under the GPLv3. An alternative to the GPLv3 would be the Creative Commons Share-Alike 3.0 license but this lacks the GPLv3's provisions against software patent abuse. At the least, using the GPLv3 gives the right to anyone who feels they can "do better" to take the standard in question, improve it, and release their new specification. Although an untested speculation, use of GPLv3 may force vendors who have made silent extensions to publish their revised specifications.

Creating the Digital Standards Organization

Institution building is costly. In July 2008, David Recordon announced the Open Web Foundation (OWF) at the O'Reilly Open Source Convention. Since then, the OWF participants have been occupied with building the organisation: collecting members, electing a board, and defining the

17 http://www.gnu.org/licenses/gpl.html

goals of the organization.[18] OWF is modelled after the Apache Foundation and has strong support from industry. However, after 18 months it is not yet ready with tools for standards developers.

When we started to build Digistan we decided to make a lightweight structure that would cost as little as possible to construct and to maintain. In practice this meant no legal entity, no formal membership, no formal elections or board. To create Digistan we used a number of email lists, based on a model we inherited from the FFII. There is an "eboard" (extended board), which takes the decisions. There is one additional list per project, plus a public discussion list. We also use a number of microsites (built using a wiki technology), which let us rapidly create homes for the various projects we wanted to start.

We started discussions in October 2007 and published our "standards for standards" in April 2009. During that two year period we also established chapters in Poland, France, Germany, and Spain, and we launched the Hague Declaration on open standards.[19]

A Reusable Legal Framework

When JPMorganChase, Red Hat, my firm (iMatix) and others created the AMQP (Advanced Message Queuing Protocol) Working Group in 2006, the legal paperwork involved in the specification process was extensive. The legal and organisational effort involved seemed disproportionate to the objectives, and in practice, the need to send contracts around for signing became the major barrier to attracting new participants. For Digistan we decided to create the lightest possible legal framework, based on the notion that each contributor licenses their contributions to others for reuse under the GPLv3. The explanatory note says[20]:

> *"This policy is specifically designed for a community with no centralizing legal entity. There are no transfers of copyright. Every contributor owns his/her work, in perpetuity, but grants a unilateral license for others to use and extend it under the conditions of the GPLv3."*

The lack of a central legal entity means no formalities for contributors except a click-through acceptance of the IP policy. The text of the policy is also short at 1,400 words, since the bulk of the legal framework is provided by the GPLv3.

The core text of the policy reads:

> *"by submission of a work to this Website in the form of a Contribution, the Contributor merges their work with the Website, with or without other Contributions, to create a Derived Work, owned by the Contributor, the Operators, and any other Contributors. The Contributor agrees irrevocably to license this Derived Work under the License and the terms of this Policy. The Contributor retains all rights over their original work."*

18 http://groups.google.com/group/open-web-discuss?hl=en%3Fhl%3Den
19 http://www.digistan.org/hague-declaration:en
20 http://spec.digistan.org/main:intellectual-property-policy

With respect to forking, the policy states:

> *"third parties may create Derived Works under the terms of the License. Derived Works may not contain misleading author, version, name of work, or endorsements. Software applications that implement specifications are not Derived Works and do not fall under the terms of the License. The use of a fragments of specification for purely illustrative purposes does not create a Derived Work."*

With respect to patents, the policy states:

> *"to the extent that a Contributor or the organization he or she represents or is sponsored by (if any) has or acquires patent claims that would be necessarily infringed by a compliant implementation of any part of any Specification, they grant a perpetual, irrevocable, non-exclusive, royalty-free, world-wide right and license to the Community with respect to their patent claims for such purpose."*

The policy assumes that a specification is developed within a microsite (website) and it classifies the entire website, along with associated email lists, irc channels, and other communication resources, as a "work". This means that when a contributor has agreed to the policy for a single website, that website can safely house many specification projects, and contributors can remix those as needed.

How COSS Works

The 1/COSS specification[21] is meant to make it easy for small teams to write, prove, and improve technical specifications. In other words, to produce small standards. It aims to be lightweight, cheap, transparent, and to fit the natural pattern by which experiments become accepted solutions, over time.I have previously said that experimentation and standardisation are two phases of the specification lifecycle. Based on experience from the AMQP specification process,[22] I defined a formal lifecycle that covered the phases of a specification as it moved from experiment to retirement:

- Raw Specifications - all new specifications are raw specifications. Changes to raw specifications can be unilateral and arbitrary. Those seeking to implement a raw specification should ask for it to be made a draft specification. Raw specifications have no contractual weight.

- Draft Specifications - when raw specifications can be demonstrated, they become draft specifications. Changes to draft specifications should be done in consultation with users. Draft specifications are contracts between the editors and implementers.

- Stable Specifications - when draft specifications are used by third parties, they become stable specifications. Changes to stable specifications should be restricted to errata and clarifications. Stable specifications are contracts between editors, implementers, and end-

21 http://www.digistan.org/spec:1
22 http://www.amqp.org

users.

- Legacy Specifications - when stable specifications are replaced by newer draft specifications, they become legacy specifications. Legacy specifications should not be changed except to indicate their replacements, if any. Legacy specifications are contracts between editors, implementers and end-users.

- Retired Specifications - when legacy specifications are no longer used in products, they become retired specifications. Retired specifications are part of the historical record. They should not be changed except to indicate their replacements, if any. Retired specifications have no contractual weight.

- Deleted Specifications - when raw or draft specifications are abandoned, they become deleted specifications. To change a deleted specification, the editor should first make it a raw specification again. Deleted specifications have no contractual weight.

What this lifecycle does is to formally define the contractual weight of any specification depending on its phase. Thus it is clear to all parties how much change they can expect, or conversely, make.

When it comes to editing a specification, we decided to restrict editing to a single person. COSS says, "*a specification has a single responsible editor, who is the only person that can edit the text and change its status. A specification may also have additional contributors who work through the editor. The editor is responsible for accurately maintaining the state of specifications and for handling all comments on the specification.*" The theory is that restricting a specification to one editor encourages a divide-and-conquer approach,i.e., a large body of work will naturally break into pieces, one per expert, creating a more layered result. Anyone has the right to fork: a fork is considered a separate specification, with its own editor.

Finally, COSS resolves natural conflicts between teams and vendors by allowing anyone to define a new specification, remixing part or all of any existing specifications as desired. There is no editorial control process except that practised by the editor of a new specification. The administrators of a domain (moderators) may choose to interfere in editorial conflicts, and may suspend or ban individuals for behaviour they consider inappropriate.

Who decides what is an authoritative specification? In traditional consortia this is done by vote. But votes are easily manipulated. COSS rejects the notion of a single authoritative specification, favouring choice and competition instead. What this means is that we ask the market to choose on the assumption that, as competing specifications (if there are multiple choices) move through from experimentation to stability, implementers will agree on the best specification. In effect we trust implementers, driven by users, to decide what is authoritative. As COSS says, "specifications have no special status except that accorded by the community."

Use cases for COSS in Real Life

In 2007-8 we tested COSS on real specifications, documenting and tracking eight small specifications (including COSS) related to AMQP.[23] Over a year or so, we were able to accurately map specifications as they moved from raw through to retired. From this experience we developed a reusable template website[24] which new workgroups could copy and use for their own specification work. This gives teams a completely functional specification tool (working wiki, legal framework, terms of use, etc.) in a few minutes.

We've since used this template website in three further specification projects:

- A set of specifications grouped under RestMS.org.[25]
- A set of specifications for the ZeroMQ[26] project, at http://rfc.zeromq.org.
- An experimental web protocol (BWTP), at http://bwtp.wikidot.com.

Today we're starting to promote COSS more widely.

Conclusions and further work

In this article we've examined the key role that free and open standards play in competition, particularly between smaller free and open source teams, and larger proprietary software businesses. We've looked at the history of standards development in the software world, and seen that traditional standards setting seems to be sub-optimal at best, and failing at worst. When powerful monopolies are threatened by new standards, they may go to great lengths to subvert those standards. When large firms work together to make new standards, they may protect these with patent pools that can keep prices inflated – legally – by many orders of magnitude.

Digistan has examined the causes of these failures and concluded that a successful free and open standard must be robust against "vendor capture" at all stages in its life-cycle. In other words, that standards can represent the economic interests of users rather than those of product suppliers. One of the key defence mechanisms against capture is "forkability," i.e. the right for anyone to create a derived specification.

Further, Digistan has developed a set of tools - a legal framework and template website - that now allow any workgroup to create a home for specifications, in just a few minutes. The Digistan legal framework uses the GPLv3 as its default license but allows other licenses to be plugged in.

On top of this legal framework, we have built a standardised process - COSS - that gives implementers peace of mind when it comes to how much change they can expect in specifications. This lifecycle formally defines specifications as raw, draft, experimental, stable, legacy, or retired.

23 http://wiki.amqp.org
24 http://spec.digistan.org/
25 http://www.restms.org/
26 http://www.zeromq.org

Each state has different contractual weight.

Finally, COSS and the underlying legal framework need no centralising organization. They allow a fully distributed peer-reviewed specification process. Authors own their own work and license it for reuse by others. Final authority is delegated to the community, i.e. implementers and users who invest in specifications.

The discussion of whether or not to allow forking has now become mainstream[27] with the HTML5 specification. This is essentially a fork of the W3C's HTML specification, yet forking is still seen as a bad thing by most respondents. We have tested these tools over the last two years in four different specification domains, and they do seem to work successfully. We expect that over time people will understand that forking of specifications is an essential freedom, and specifications will move more and more to share-alike licenses. Just as with software, the license defines a community, and the rules on remixing the work of others play a significant role in growing the community.

Acknowledgements

The Digistan eboard (founders) email list was instrumental in developing Digistan's legal framework, IP policy, and template web sites. Mark Webbink and Graeme West provided very helpful review comments and improvements to the article.

About the author

__Pieter Hintjens__ is an expert in the creation of collaborative communities. He has 30 years of experience in the software business, starting his first company at the age of 18. He wrote his first GPL application, Libero, in 1991. He is past president of the FFII, which works for freedom of the software economy. As FFII president he organized the European Patent Conferences (EUPACO), the Digital Standards Organization, the European Software Market Association (Esoma), the Campaign for Ethical Patents, the NoOOXML.org campaign, and many smaller projects. He is also founder and CEO of iMatix Corporation, which builds the worlds' fastest free messaging software, ZeroMQ. He was the lead editor for AMQP (the Advanced Message Queuing Protocol), and for the RestMS web messaging protocol.

27 http://www.w3.org/2010/Talks/doclicense-20100323/#%281%29

Call for Papers

The International Free and Open Source Software Law Review (IFOSS L. Rev. or IFOSSLR) is a collaborative legal publication aiming to increase knowledge and understanding among lawyers about Free and Open Source Software issues. It is seeking submissions from qualified authors in a variety of research areas. The topics covered by the publication include copyright, licence implementation, licence interpretation, patents applicable to software and business methods, standards applicable to software, case law, statutory changes, license enforcement, competition law applicable to software, economics analysis, business models and due diligence. Issue 4 of IFOSS L. Rev. will be released on December 7th 2010. To be considered for inclusion in this issue articles must be submitted by October 1st.

Full-Length Articles

IFOSS L. Rev. accepts full-length researched articles on legal issues of interest to the publication's readership. All accepted submissions are peer-reviewed. No fixed length is prescribed, but 5000 - 15000 words is a useful indicative range.

Legislative Review

This section contains briefs on recent or upcoming changes to legislation relevant to Free and Open Source Software. Articles discuss the main implications of the legislative change, set out the context of prior law, and indicate how governmental policy and/or international law will affect implementation. All accepted submissions are peer-reviewed. No fixed length is prescribed, but 1500 - 3000 words is a useful indicative range.

Case Law Reports

The focus of this section is reports on recent developments in case law that offer commentary on the legal and policy implications of the decisions. Articles are peer-reviewed where appropriate. No fixed length is prescribed, but 500 - 3000 words is a useful indicative range.

Book Reviews

IFOSS L. Rev. accepts reviews of recent literature relevant to Free and Open Source Software. Book reviews are normally 500 - 2000 words in length, and should critically analyse the main arguments of the literature under examination. Further, they should indicate the reviewer's assessment of the literature's usefulness to IFOSS L. Rev's readership, its novelty of approach, its factual accuracy and its relevance to current debates.

Tech Watch

IFOSS L. Rev. prides itself on outstanding industry awareness and cross-disciplinary reach. Tech Watch is therefore offered as a forum to set contemporary technical and social issues in the Free and Open Source community into their legal context. These discussions may form the basis of subsequent detailed examination by other contributors. Tech Watch columns take a in short essay form, and are expected to be between 500 and 2000 words in length. They are usually written by non-lawyers.

Platform

IFOSS L. Rev. accepts articles for publication in the Platform section from all qualified authors. Because of the nature of this section as providing, as the name suggests, a platform for the expression of a personal view, submissions are reviewed by the full Editorial Committee, and both the detail and spirit of the standards of this publication are expected to be adhered to. In addition, because of its nature as a venue for reasoned opinion, due consideration for respect and professional courtesy are essential qualities for any article intended for publication in this section. Platform articles may be of any length.

Submission process

Submissions are now being sought for publication in 2010 and beyond. You can contribute a paper by following the instructions online at:

http://www.ifosslr.org/index.php/ifosslr/about/submissions

Volume 02} Issue 01} June 2010

“As a consultant and commentator on current IP developments, and as editor of a leading IP journal, I’m pleased to welcome IFOSSLR’s contribution to debate on a most important issue -- and I’m highly impressed with the standard of its content”

-- Jeremy Phillips
Intellectual Property Consultant, Olswang LLP; editor, Journal of Intellectual Property Law & Practice; blogmeister, IPKat weblog.

Thanks to our Sponsors
IFOSS L. Rev. is published by its Editorial Committee. The Committee gratefully acknowledges the sponsorship of its many supporters and donors, including:

nlnet FOUNDATION mozilla FOUNDATION

Available online at: http://www.ifosslr.org