Identifying and Controlling Legal Risks of Open Source Software in a Software Company

Juho Sivonen

Department of Commercial Law

Hanken School of Economics

Helsinki

2014

HANKEN SCHOOL OF ECONOMICS

Department of: Department of Commercial Law	**Type of work: Thesis**
Author: Juho Sivonen	**Date: 5.11.2014**
Title of thesis: **Identifying and Controlling Legal Risks of Open Source Software in a Software Company**	
Abstract: The use of open source software has increased over the last decade and now almost all software companies are using open source software in their business in one way or another. This thesis aims at identifying and analyzing the underlying legal risks involved with using open source software from a software company's point of view. The legal issues are put into a general IPR context from the point of view of European legislation. Although European legislation is not a homogenous concept, the general nature of legislation instruments (e.g. Regulations, Directives) have harmonized legislation in the European Union on a fundamental level. Scarcity of European court precedents compels to make comparisons to US cases and legal literature. Moreover, this thesis aims to provide some general guidelines for software companies on how to address and minimize identified legal risks and to implement efficient policies in order to do so.	
Keywords: Intellectual property, software, open source, licensing, license compliance, open source policy	

CONTENTS

1 INTRODUCTION

1.1 Introduction to the Topic

Open source is a movement, an ideology and a business model. It is also a brave new point of view to software licensing which has fundamentally changed the way software companies do business. However, at the onset, open source software was considered something exclusive to computer nerds. From the very beginning of the idea of open source, first launched by Richard Stallman in the late 1980s, open source software has come a long way from being a politically motivated movement and a critique towards commercial or proprietary software (i.e. closed source software). What might not have been foreseeable in the early 2000s has become business as usual. In 2014, it is safe to state that open source software is ubiquitous and more importantly, a widely recognized and respected way of conducting business in the software industry. Examples are abundant as the biggest players in the software industry, such as Google, Facebook, Yahoo and Netflix all rely heavily on open source software.[1] According to a Gartner survey, 85 per cent of software companies use free and open source software and the remaining 15 per cent are expecting to use it in the next 12 months.[2]

This rapid adoption of a new way of looking at software and related intellectual property rights ("IPRs") is, if not a complete shift in the software business paradigm, at least part of the mega trends that have taken place in the software industry in the past few years.[3] As other mega trends may be trendier topics and have taken over the majority of the academic and business related discussion, the proliferation of open source software has quietly crept from the messy tables of software engineers and coders to the frontlines and cores of software companies' product lines.

If software engineers and other enthusiasts were the early and vocal adopters of open source ideology, the same cannot be said for software business executives. In his infamous statement, Microsoft's CEO Steve Ballmer characterized the open source

---

[1] See e.g. http://www.itviikko.fi/ratkaisut/2014/06/12/nain-linuxilla-tehdaan-rahaa/20148284/7, last visited 12.6.2014.

[2] See Kemp 2009 p. 61.

[3] Forrester Webinar "Megatrends Shaping Software Development", 22.4.2013. Other past or on-going mega-trends in the software industry worth mentioning are for example consumerization, cloudification and big data. Description available at https://www.forrester.com/Megatrends+Shaping+Software+Development/-/E-WEB13823, last visited 28.10.2014.

ideology to be inherently detrimental to business.[4] Ballmer's characterization of the open source ideology manifestly illustrates the change of mind and adoption of new business models and opportunities involved with the adoption of open source software. To this date, although Ballmer is no longer CEO, Microsoft actually relies heavily on open source. For instance, its cloud based Azure-service relies for its main parts on open source software. Microsoft has even adopted its own open source license.[5]

I would go as far as to interpret that the adoption of open source software has facilitated in part the shift in software companies' focus from physical products to services run from the cloud[6]. This shift from products to services is ubiquitous and can be seen everywhere in the industry. For example, the amount of data running through data centers is estimated to grow from 2.6 zetabytes (measured 2012) to 7.8 zetabytes in 2017.[7] This exponential growth illustrates this on-going shift to cloud services.

This shift has resulted in and will bring more changes as to how software companies conduct their business. Moreover, companies may have to re-assess their internal guidance and policies (such as IP, innovation, licensing and software development policies) in order to better adopt the benefits of open source technologies. In order to achieve this, a company should carefully identify and acknowledge the various legal risks and implications involved in using open source software. However, the myriad number of open source software licenses might become a head-ache for companies' legal departments as the legal status of open source licenses is ambiguous to state the least. Companies should understand that using open source software is always subject to complying with the license obligations of each license. As these obligations are fundamentally different from commercial license obligations, companies need to pay special attention in their quest of using open source software. It is perfectly logical that the company's own legal department and outside law firms become involved in the identification and interpretation of legal risks involved in the open source licenses in order to assess whether the chosen open source licenses qualify and meet the expected benefits of the business case. As some of the open source software licenses are far more simpler and easier to comprehend than others, and as companies' own lawyers or

---

[4] http://news.cnet.com/2100-1001-268520.html and http://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer, last visited 27.2.2014.

[5] http://opensource.org/licenses/MS-PL. Last visited 27.2.2014.

[6] Cloud computing refers to computing in which large groups of remote servers are networked to allow the centralized data storage, and online access to computer services or resources. For more on cloud computing, see e.g. http://en.wikipedia.org/wiki/Cloud_computing#Services, last visited 28.10.2014.

[7] David Aaroe at Bird & Bird Conference 28.1.2014, topic of speech "Datacenter Growth Trends".

outside attorneys cannot be involved in every single case when licenses are considered, it is sensible for a software company to introduce an open source software policy where the boundaries and use scenarios of most common licenses are presented.

1.2 Research Questions

In this thesis I aim to identify main legal risks for software companies from a general intellectual property and license (or contractual) point of view and to provide practical guidelines of facts which should be taken into consideration when creating open source policies. The legal risks referred to above are limited to IPR infringement and contractual and license breach. It has to be noted that software companies are very different from each other and their products and hence also their business models vary greatly. Even though this is the case, software companies have more things in common regarding the use of open source software than they might have business wise.

For instance, one software company might be leveraging benefits of open source software by implementing open source components in its end-user products whereas another company might simply try to benefit from open source software in its internal and non-visible operations, such as its backend systems[8]. Both companies need to, however, understand that both use scenarios have risks with different legal implications. On the one hand, a company using open source software in its end-user products needs to understand how it can legally maximize its rights granted by specific open source licenses and to, furthermore, understand how such open source licenses are legally and contractually intertwined and how they may be used together with traditional commercial licenses. On the other hand, the other company will need to tackle similar issues but from another point of view as it may need to consider possible future business models although its current logic may require only a loose guideline to the use of open source software. As an example, a company might offer a traditional commercial licensing model of software that is being delivered as a service from its closed backend systems.

However, as the change in software companies' business models have gone towards offering products as services rather than physical products and software licenses, the legal premises of using open source software could change dramatically. Where the company may have blocked all access to its closed backend systems before, as the same product is offered as a service from the companies servers running the same

[8] Definition of backend system available at e.g. http://en.wikipedia.org/wiki/Front_and_back_ends and http://www.techopedia.com/definition/1405/back-end-system, both last visited 28.10.2014.

closed backend systems, such business model might be considered as distribution. Those backends may thus be subject to reciprocity obligations arising from the licenses which in earlier business models might not have been flagged as potential triggers for such license obligations.[9]

The main point of interests of this thesis is the identification and assessment of legal risks involved in using open source software from a software company's point of view. The definition of "software company" used in this thesis is flexible and broad. A software company is to include companies which deliver software products to consumers or other companies for profit. Furthermore, the central research goal of this thesis is to connect the identified and assessed risks in open source with practices that may minimize the risks without losing the expected benefits of using open source licensing.

The second chapter explains open source in order to better connect its expected benefits and known business models to the context of this thesis. The third chapter focuses on identifying and assessing legal risks in open source software and to categorizing them to provide recommendations on how to introduce an open source policy. The chapter will start with a general part on copyright protection for copyright and will then move on to more detailed analysis. Moreover, the main legal concepts involved in open source licenses are put in a more general IPR context. The fourth chapter shall include separate sections on which elements are crucial when planning an open source policy and how a company should incentivize its personnel to comply with the policy. The final chapter draws together the conclusions of this thesis.

The main sub research question is why should a software company have an open source policy in the first place? Although many lawyers recommend it, for various reasons, and it might at first seem trivial or non-scientific, the very rationale behind open source policies as tools in order to police software or product development in a software company has been challenged by some commentators.[10]

Moreover, I aim to conclude what kind of a risk management tool an open source policy is, regarding, especially, in terms of legal risks and enforcement. I will attempt to identify

[9] For this purpose, see terms "propagate" and "convey" in the General Public License, Version 3. Available at http://opensource.org/licenses/GPL-3.0.
[10] See e.g. Meeker 2008 p 119-120.

and analyze specific open source license obligations which might be problematic both from a compliance and risk point of view.

However, before being able to properly address the first research question, the legal risks involved in open source licensing need to first be identified and categorized. In this context, only a few key legal concepts will be focused on. These are copyleft, derivative works and distribution which will all be looked in more detail. Furthermore, the above concepts are being put into a general copyright context.

In this thesis, I will focus on inbound open source licensing, i.e. open source licenses which a software company acquires from outside in order to use internally or to distribute as such or as incorporated into its own software and services, whether in unmodified or modified form. This distinction shall, therefore, exclude outbound licensing, i.e. where a company uses open source licensing for third parties in order to gain competitive advantage over its competitors. However, business models discussed in this thesis may have elements from both inbound and outbound licensing and in practice the division between the two is not straightforward as a company may well use acquire an open source component (inbound licensing) and then distribute it, in modified or unmodified form (outbound licensing).

Furthermore, I will attempt to identify and analyze the crucial and most common elements of an open source policy from a legal point of view. For this purpose I will ascertain why open source software is not only a beneficial part in a software company's business plan but furthermore why using open source software will yield distinct advantages compared to traditional commercial software. In order to facilitate understanding of the foregoing, the most common open source licenses and the rationale of open source related business models will be discussed in some detail prior to presenting and assessing legal risks and their positioning in an open source policy.

However, although using open source software may have many benefits, software companies should realize that using open source software without proper automation related to guidance and compliance, it might divert business and might consume technical and legal resources, thus adding to the true cost of software which might have been initially acquired without a fee.[11] For this purpose, chapter four focuses on providing

[11] http://techcrunch.com/2012/12/14/open-source-software-compliance-basics-and-best-practices/, last visited 30.9.2014.

some guidance on what kind of elements are needed in a company's open source policy in order to best capture the value of open source but simultaneously tackling the most common legal risks associated with open source.

1.3 On the Method and Sources

Although this thesis is for its most parts a legal study, and as companies do not operate in a legal vacuum, it does have some business and economics related aspects as well. For this purpose, the business purpose is always attempted to be kept in mind. However, as this is mostly a legal study, the analysis is mainly done from a legal point of view, and therefore normal sources for legal studies, such as legislation, court praxis and legal literature are used. This thesis will also contain descriptions and analyses of some of the main open source licenses used in the software industry.

The thesis is written from a European point of view, with weight on European legislation and contractual interpretation, but will also, for reasons of scarce availability of purely European legal literature on open source licenses, include Anglo-American viewpoint on the same matters, for comparative insights.

Legislation does not play a key role in this thesis, as the main focus will be on the contents and analyses on the actual obligations and enforceable terms of open source licenses. The conclusion from such analyses are used in order to attempt to establish rules and procedures for setting up a working and useful open source policy. In this context, legal literature, especially American literature, will play an important part in understanding and establishing why certain matters need to be carefully looked into before execution. As most of the information and literature regarding open source in general is available online, it is therefore only logical that the majority of sources for this thesis are articles and other shorter material. It should also be noted that legal literature (other articles) is scarce on matter.

Certain questions regarding the validity and borders of open source licenses arise from contractual background whereas some questions are more related to intellectual property and the separation of these issues may be unpractical taking into consideration the goals of this thesis as well as for a company assessing the very same issues. A practical example of a contractual issue regarding open source software is assessing the potential risks of non-compliance with open source license terms. As has been pointed out by numerous commentators and confirmed by courts in different jurisdictions, open

source licenses are enforceable as contracts and bear legal obligations in cases of non-compliance.[12] What these legal consequences actually are, is to be discussed later in more detail.

Furthermore, the contractual obligations lead, in practice, to further questions, more specifically and traditionally related to intellectual property law. The example about contractual issues presented briefly above will spur debate inside an organization planning an open source policy on when and how specific clauses of open source licenses are triggered. For instance, an issue often referred to in legal literature as the "border dispute" is very relevant for a company using or planning to use a particular open source license in order to assess whether its own products are to be considered as derivative works of the open source software or vice versa.[13] This is where national laws on intellectual property come into play as open source licenses do not generally have provisions on governing law.

In some cases, the issue might be both contractual and one that relates to intellectual property law as an open source software license might state that all derivative works based on the program shall be governed and licensed under the same license. This might be simple in theory but might prove to be burdensome to assess in practice as the licensee will have to assess, both based on the license terms and relevant copyright legislation, how a derivative work (i.e. a work based on the program) is to be defined. This assessment is crucial to companies whose whole business logic is based on laying a strong ground for its intellectual property rights.

As an outcome of the previously stated, the sources shall vary from European to American legal literature. However, possible outcomes of this thesis shall be presented from a European perspective which means that all problems and solutions might not be aligned in the US and Europe. For instance, the American approach and rationale for having an open source policy might have more to do with limiting its (and its officers') liability than to actually spur and incentivize the use of open source software in order to create new innovations.[14] The challenge in this approach is to understand the differences in reasoning and finding the underlying reasons in order to implement a working open source policy.

---

[12] See e.g. Van den Brande et al 2014 for country specific open source litigation.
[13] See e.g. Meeker p. 138.
[14] As an example, see e.g. Meeker p. 119-122.

Due to its nature, another key source for this thesis are all relevant open source licenses and their actual rights and obligations and other key clauses. As there is an abundance of open source licenses available for different purposes, only the most relevant ones from a commercial point of view will be discussed in more detailed whereas the others might be simply used as an example or mentioned for other reasons.

Even though not the most conventional topic, this thesis is a legal study, and therefore the methodology of traditional legal studies shall at least be attempted, although it stands to reason that the contents and possible outcomes of this thesis are not something that in traditional legal studies would be considered to fall within "as systematizing and interpreting existing laws or legal doctrine" which are often regarded as the traditional methods for legal research.[15] In this sense, the aim of this thesis is to apply legal doctrine and business logic in order to better understand the legal risks involved in open source but to also better understand and to leverage their business potential.

2. What Exactly is Open Source Software?

2.1 Defining Open Source Software

The starting point for each and every legal article concerning open source software seems to be the defining of the term "open source software" followed with a brief historical background. As this thesis doesn't offer any exception to that rule, this study starts by summarizing twenty something years of open source history.

The open source ideology, first launched by Richard Stallman in 1983, was at some point a clear political and social ideology not strictly limited to software. Earlier movements had existed, but Stallman created the free software movement[16] which ultimately defined what free software is and what kind of software qualified as such. Earlier similar freedoms and declarations had been made but merely for practical or scientific reasons. Stallman defined free software as "software which all users are free to use, study, modify and redistribute.[17] The often used analogy is that software should be free as in freedom, not free as in free beer.[18]

---

[15] Further on traditional legal research methods, see e.g. Aarnio p. 48.
[16] http://en.wikipedia.org/wiki/Free_software_movement, last visited 2.4.2014.
[17] http://www.gnu.org/philosophy/free-sw.html and http://opensource.org/osd-annotated, last visited 2.4.2014.
[18] Stallman 1999 p. 12 and http://www.gnu.org/philosophy/free-sw.html, last visited 28.10.2014.

In 1985 Stallman published the “GNU Manifesto”[19] along with his definition of free software, the copyleft ideology[20] and, in addition, the related GNU Project which aimed at creating an operating system free from any intellectual property or commercial contractual constraints. It is told that the spark for this project came from a printer which Stallman was able to fix but was not allowed to due to source code being owned and withheld by the company manufacturing the printer.[21] According to his manifesto, Stallman wanted users of free software not to need to ask for permission in order to make changes in source code or having to agree to restrictive proprietary licenses with deliberate unavailability of source code. In 1989, the first version of the GNU General Public License was published, subsequent versions of which have become one of the most used and well-known license types globally.

The best known open source project is probably Linux, which was started by Linus Torvalds while working at the University of Helsinki. Torvalds was unhappy with current operating systems and started working on his own operating system, aiming to make a kernel[22] which would be usable and accessible for everyone to contribute.[23] The Linux kernel was released in source code format in 1991. Although not initially licensed under Stallman’s GNU General Public License, Torvalds re-licensed the Linux kernel under the GNU General Public License later in 1992. As the kernel was licensed under Stallman’s free license, it attracted the attention of volunteer programmers all over the world resulting in the combining of the Linux kernel and Stallman’s GNU operating system, making it the first operating system which qualified entirely as free software.

To this day, multiple Linux based products and distributions have been launched, both commercial and open source. For instance, Red Hat Inc launched its commercial Linux distribution for consumers, called the “Red Hat Commercial Linux”, in 1994.[24] It was discontinued in 2004 although Red Hat has since then thrived with its business-related Linux distributions. Popular open source Linux distributions include Debian, Fedora and Ubuntu.

---

[19] http://en.wikipedia.org/wiki/GNU_Manifesto, last visited 2.4.2014.
[20] http://en.wikipedia.org/wiki/Copyleft, last visited 2.4.2014.
[21] Transcript of Richard M. Stallman's speech, "Free Software: Freedom and Cooperation", New York University on 29.5.2001. Available at: http://www.gnu.org/events/rms-nyu-2001-transcript.txt, last visited 2.4. 2014.
[22] A kernel is the central component in most operating systems, for more see e.g. http://en.wikipedia.org/wiki/Kernel_%28operating_system%29, last visited 28.10.2014.
[23] http://eu.conecta.it/paper/brief_history_open_source.html, last visited 2.4.2014.
[24] http://en.wikipedia.org/wiki/Red_Hat_Linux, last visited 2.4.2014.

Although Stallman was a vocal proponent and the man behind the first free software definition, it was a group of other people who came up with the term “open source” in a free software movement strategy session, inspired by Netscape Communications releasing the source code of their internet browser software in 1998. The very same source code is the basis for e.g. the popular Mozilla Firefox browser.

It has been construed that the free software movement was too political in the eyes of the software industry and that has been speculated to be the main reason for the adoption of the term “open source”. Eric S. Raymond and other like-minded software engineers wanted to diverge from the politically loaded free software movement and bring free software closer to the principles and economic benefits of commercial software. Raymond and his colleagues rebranded the term “free software” to “open source” by which they looked, not only to make the term more alluring to business, but also to emphasize that sharing code is in the midst of the ideology.[25]

Since then, the open source definition has been introduced, through the Open Source Initiative which was founded in 1998, to its now current version (v 1.9) which comprises of ten different criteria, such as free distribution, open source code and neutrality of technology it is applied to.[26] Although the official open source definition is available, the term is these days used for anything contrary to commercial or proprietary software. It should be noted that even the terms “commercial” or “proprietary” are not without ambiguity as companies have since the late 1990s adopted hybrid business models which include combination of all types of licenses. It has to be stressed that open source does not equal public domain, on the contrary, the various license obligations keep companies cautious and even suspicious about what certain provisions actually mean. In a hybrid business model software companies merge open source and proprietary software into each other.[27]

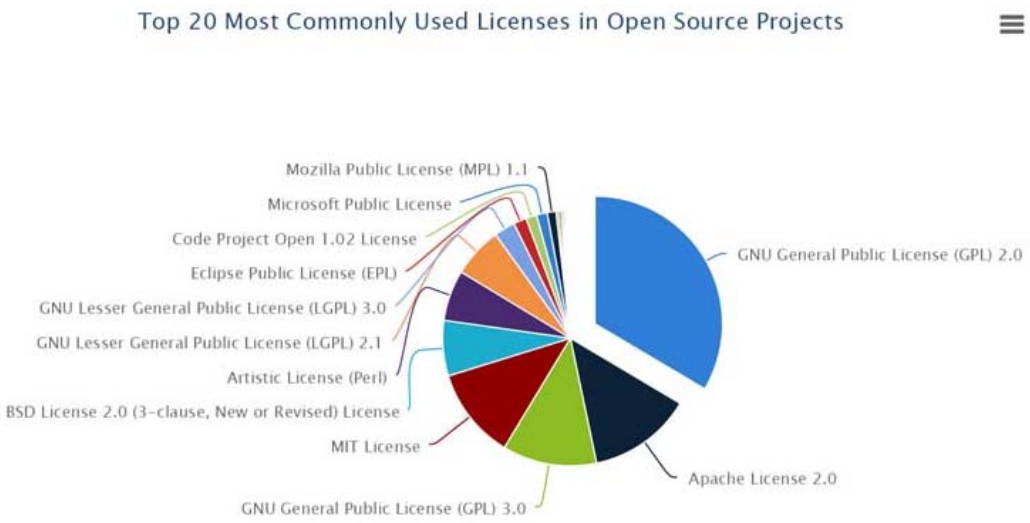
2.2 Most Common Open Source Licenses

From the early days we have seen an abundance of open source licenses. According to a Black Duck study from 2014 the number of on-going open source projects will reach something closer to two million during 2015.[28] According to Black Duck, as can be seen

---

[25] Raymond 2001 p. 10.
[26] http://opensource.org/osd-annotated, last visited 2.4.2014.
[27] Se e.g. Ballardini 2012 p. 10.
[28] Black Duck Inc: “Top 10 Open Source Legal Developments in 2013”, Webinar 3.4.2014. Available at: http://www.blackducksoftware.com/resources/webinars#251, last visited 8.4.2014.

from the below figure, the General Public License (GPL) 2.0 is the most common license with nearly 33 per cent share of all open source license use, whereas its successor, the GPL v. 3.0 comes far behind with less than a 12 per cent share, albeit its use seems to be growing rapidly as companies have become more accustomed to it. Other popular licenses include Apache 2.0 and other permissive licenses, such as the MIT license, Artistic License and the BSD 2.0 license.[29]



Open source licenses differ from traditional licenses in the way that they govern simultaneously distribution and end-user terms.[30] Open source licenses are often categorized into two rough categories. The first one comprises of permissive licenses such as the ones referred to above. Despite being among the first ones to be adopted, their popularity clearly indicates that there is a need for different types of open source licenses. Software licensed under permissive licenses usually tend to be useful, often work saving libraries or large, on-going projects that are in need of constant change. One inevitable argument for using permissive licenses is the hassle-free, ease of use as permissive licenses are often short and clear about their use obligations as they pose minimal restrictions on use. Often the only obligation is to keep the original copyright notice in tact. For instance, BSD's popular 3-clause license only consists of three contractual obligations.[31]

---

[29] Above figure and listing of most popular open source licenses available at: https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses, last visited 8.4.2014.
[30] Van Holst 2013 p. 6. Van Holst calls open source licenses dual-purpose licenses.
[31] See http://opensource.org/licenses/BSD-3-Clause, last visited 9.4.2014.

The second category of open source licenses is often referred to as strong copyleft[32] licenses as they, by default, require reciprocity in terms of disclosing changes made to the source code. The second category can be further split into two subcategories as it has been established in legal literature that copyleft licenses are to be distinguished by their nature whether they have a "viral effect" or whether they simply require that their license terms remain unchanged if software under such licensed is combined with other software ("share alike").[33]

Share alike licenses are often permissive when it comes to combining them with software works covered by other licenses. Characteristically, this type of licenses are approved more easily for business use in companies than licenses with a strong viral effect as they allow combining open source components with e.g. proprietary software.

By viral effect, in this context, is meant that viral licenses require that if a work covered by it is combined with another work, the outcome (i.e. the combined work) is covered by the copyleft license. As the legal effect of strong copyleft are yet to be tested in court and hence their enforceability is somewhat uncertain, they pose a clear risk to software companies as they strive to comply and work around these kinds of viral licenses in order to avoid an "open source contamination" of their proprietary software. Some strong viral licenses, such as the Affero General Public License (AGPL) go even further by stating that the viral effect covers also use over a network. This is probably the main reason why the AGPL remains one the most avoided and red-flagged open source licenses in software companies. Other licenses with a strong viral effect are e.g. GPL v3.0 and the Open Software License (OSL).

It can be argued that open source licenses with a strong viral effect are not as free as their advocates and proponents have vocally been claiming as they impose very specific and sometimes even unreasonable obligations for their use.[34] In this case it may be more favourable in terms of security and predictability for a software company to procure a commercial alternative instead of an open source alternative.

---

[32] By copyleft is usually referred to a licensors intent to guarantee that all licensees get a right to copy, modify and distribute the original work. The copyleft obligations of an open source license are almost always triggered by distribution. For a more detailed definition, see e.g. Deek-Mc Hugh 2008 p. 338 and Meeker 2007 p. 23 and 277.

[33] On categorization, see e.g. Välimäki 2009, p. 205-208.

[34] Välimäki 2009, p. 208.

The most problematic legal provisions and obligations of open source licenses, namely licenses with a strong viral effect, are discussed in greater detail from a software company's point of view in chapter 3.

2.3 Why Open Source? Benefits of Open Source and Introduction to Business Models

The rationale for using open source software business should be addressed from an economics point of view as all business related decisions ultimately come down to whether it affects the bottom line on the statement of earnings.

This sub-chapter aims to merely point out the different rationales for using open source software, whether it is a matter of simply cutting costs on duplicate work or license costs or whether a company's whole business model leans on reaping benefits from open source software. Secondly, some open source business models are introduced.

Setting other aspects aside, there isn't a clear-cut division between "open source companies" and "proprietary software companies" as almost all major companies rely on open source software at least to some extent. Small and midsize companies having seen the proliferation of open source software, not only as an ideology, but as a technological and commercial innovation, have become used to incorporating the open source ideology into their business models.[35] According to Mustonen, the mere existence of open source business models affects competition in a positive way as new business models force established software companies to improve their products in order not to fall behind.[36]

Open source, along with cloudification[37], may have changed the thinking and underlying business logic of many traditional software companies as software tends to these days be offered as a service rather than a restricted license. With the freedom of not having to install and to maintain multiple platforms, software companies have been able to offer a clearer service to their customers. According to Asay, the reason why open source software has become so relevant in software and technology industry is that "the industry

---

[35] Perens 2008 p. 37.
[36] Mustonen 2003 p. 119.
[37] Cloudification means bringing cloud computing to the masses, for further details see e.g. http://www.computerworld.com/article/2471400/cloud-computing/cloudification--bringing-cloud-computing-to-the-masses.html and http://en.wiktionary.org/wiki/cloudification, both last visited 28.10.2014.

is in the midst of a tectonic shift in how software is delivered; a shift from the value in the bits themselves to value in the service around the bits".[38]

From an economics point of view the main goal of open source software is that it offsets monopoly rights traditionally connected to intellectual property through copyleft obligations.[39] One might even point to the phenomenon known in economics as the "tragedy of the anti-commons" which suggests that under-use of private goods that are controlled by more than one right holder is a bad choice from an economics point of view.[40]

According to Dolmans, it makes far more economic sense to apply common standards for the same kind of functions and products which are able to interoperate with each other.[41] As an example, the smart phone contains thousands of patents and thus represents different technologies which can be done in a great number of manners, however, the parties involved have chosen to create them in a way which ensures they work together as they would be worthless on their own. Furthermore, non-interoperability monopolizes innovation and results in company-specific technology.[42]

As the original ideology of the open source movement aimed to abolish all intellectual property related to software as they were seen as a limitation on making software widely available , but still wanted to create incentives to prevent economic freeriding by making minor changes to open source software and then releasing it as their own, strong copyleft obligations envisaged in licenses such as the General Public License and Affero General Public License were introduced.[43] These copyleft obligations are in place to enforce the nature and ideology of open source through contractual obligations.

The rationale for using open source software may vary greatly between companies. For this purpose, use scenarios can be divided into internal and external use. Furthermore, some software companies rely on hybrid business models, such as dual licensing where the company may offer the same software or service through a free open source license or, subject to payment and less restrictive license terms, a commercial version of the

[38] Asay 2010 p. 197.
[39] Kumar 2006 p. 3.
[40] Heller – Eisenberg 1998 p. 1.
[41] Dolmans 2010 p. 119.
[42] Dolmans 2010 p. 119.
[43] Evans et al 2008 p. 74.

same software which can be e.g. used commercially without any typical open source software license term obligations, such as the obligation to share any derivative works.

Before going further into details on the above advantages, it should be stated that one common argument for the quality of open source software project often relate back to why programmers contribute as greatly to open source projects as they do. One reason for this is publicity. As open source code is more scrutinized than code for commercial software in general, the quality for the code, in order for the programmer to actually attract acclaim and peer recognition, has to be high. This is probably why so many senior software developers contribute to open source projects. Another reason for this might well be the abundance of corporate participation in open source projects.[44] In terms of economics, according to Evans, if a critical amount of software developers consider a particular piece of software potentially useful, they all contribute and pool their talents in order to develop the software.[45]

On the one hand, the shift towards growing corporate involvement in open source projects stems from the fact that gifted software developers require that they are free to work on open source projects not only on their free time, but also during working hours. This might be a result from generation of software developers having grown up by tinkering with different open source component freely available online.

On the other hand, employers have seized their opportunity for this kind of demand by having their software developers work in open source projects which are in the best interest for the company. For instance, many open source projects, such as the Linux Foundation, are funded and even partly operated by major software companies.[46] Software companies have large economic incentives to participate in open source projects that are closely linked to their own products and services. As an example, Linux presented as a ready solution for IBM in order to integrate all of its servers without having to develop, and furthermore, provide support to, a commercial software platform of its own.[47] Furthermore, Facebook participates in hundreds of different open source projects, the company has over 200 on-going open source projects on collaboration platform GitHub alone.[48] Through donations and man hours offered through its developers,

---

[44] Evans et al 2008 p. 110-111.
[45] Evans et al 2008 p. 76-77.
[46] See e.g. Mann 2006 p. 25 and http://www.linuxfoundation.org/about/members, last visited 17.4.2014.
[47] Evans et al 2008 p. 77.
[48] http://opensource.com/life/14/9/interview-james-pearce-facebook, last visited 1.10.2014.

software companies may be able to steer the development of individual open source projects into a commercially favourable direction.

Besides the vast availability and quality of code involved in open source projects, the advantages of open source software are manifold and debatable. For the purpose of simplification, six main categories of advantages of open source software are identified; *reliability, stability, auditability, cost, flexibility and support*. These advantages are looked into in more detail below. The above categorization is based on a UK Government analysis in order to assess the impact of open source software.[49]

2.3.1 Categorization of Open Source Benefits

Reliability

Reliability, in this context, refers to the speed at which known bugs or severe defects are fixed within the open source community. As the source code is publicly available, the code is scrutinized by a far more heterogeneous crowd compared to commercial software projects. This amounts, not only to a fewer amount of bugs and defects, but also to the fact that they are fixed in a timely manner from their initial detection.

It also means that developers who discover bugs or defects usually report them back to the community without delay. This community-led process usually works much faster than with commercial software as in that case the end-user is completely dependent on whether and when the software vendor decides to issue an update. It might well be that the vendor doesn't consider the issuance of an update to be commercially viable, especially in cases where the defect is critical but only affects a minority of its customers. Moreover, a software vendor's internal processes for issuing critical updates may not only be non-transparent, but also slow and inefficient.

Stability

Stability might mean different things to different companies, but from a customer point of view the hassle of upgrading a perfectly working piece of software for merely entertaining the software vendor's financial motives might prove to be frustrating. For example, Microsoft end-of-lifed support for its old Windows XP operating system (launched in 2001) in April 2014, and although the announcement was made several years ago,

[49] Peeling & Satchell, QinetiQ, 2001. This document was commissioned by UK Government to analyze the impact of Open Source Software from a Government Policy perspective. Available at: http://www.govtalk.gov.uk/documents/QinetiQ_OSS_rep.pdf, last visited 17.4.2014.

hundreds of millions of computers became susceptible to attacks as Microsoft wouldn't provide any e.g. security updates from the end-of-life date.[50] It has been widely speculated that the reason for such a big media campaign on the end for support for the operating system is that Microsoft was hoping clients would make the shift from the obsolete Windows XP to one of its current operating systems, such as Windows 8 which has been struggling in sales since its launch in 2013.[51] Although this mere example might not fully contemplate and take into consideration the complexity of issuing new products and changes in technology, it illustrates the difference between market-oriented commercial software and open source software in terms of stability.

Had Windows XP been an open source product, the community would have continued to support the platform even beyond Microsoft's final end-of-life date. Reasons why support for major open source projects might be end-of-lifed have more to do with the evolution of technology and compatibility standards than with breaking support in order to push revised products to the market. A choice to use open source software solutions might shelter the customer or end-user, at least to some extent, from costly regular upgrades. If needed, as the source code is available to everyone, a company might even support its own software if support for the specific software becomes universally unavailable.

Auditability

Auditability is, according to the UK Governments study, an often overlooked and rarely understood quality in open source software.[52] As the source code is public, third party inspection of the software is an easy option. Contrary to commercial software, where the customer rarely has the means to actually audit the software on a source code level. Naturally, other kinds of tests do exist, but none of them are completely transparent as opposed to disclosing the source code.

Software companies do often reassure their customers on e.g. the security of the software by providing quality certificates in order to prove to the customer that the software is sufficiently secure. However, the customer doesn't know the details of the audit and what kind of shortcomings the software might have. In addition, vendors rarely

---

[50] https://www.microsoft.com/en-us/windows/enterprise/end-of-support.aspx, last visited 17.4.2014.
[51] http://windows.microsoft.com/en-us/windows/history#T1=era10, last visited 17.4.2014. On struggling sales see e.g. http://online.wsj.com/news/articles/SB10001424127887323826804578468823595533416, last visited 17.4.2014.
[52] Peeling & Satchell, QinetiQ, 2001.

agree to contractually maintain the level of e.g. security as provided by the certificate, as their contents might change. Furthermore, vendors often limit the warranty of the software in a way which rarely results in compensation even if the software is non-conformant with its documentation. According to the study, open source software have probably paved the way for software audits even for commercial software.

Furthermore, as the source code is publically available, a peer evaluation has become the prevailing system for open source software. Should commercial software be disposed to similar scrutiny, it can only be imagined how many backdoors and security defects were to be found.

Cost

Cost should never be undermined as a reason to implement or use open source software as most current open source software and large open source projects are available free of charge.[53] Use of open source software may result in significant cost saving, both in in terms of license and development cost.[54] However, cost is probably the most misunderstood and misstated arguments for using open source software. As stated above, open source software is free as in freedom but not necessarily free as in free beer (libre vs gratis).[55] Although the term free in open source software as a concept refers to the freedom to operate rather than free of charge, most open source software are, however, distributed free of charge.

However, the cost of procuring software shouldn't be the sole factor for using open source software, but as the cost is zero, open source software may just have the upper hand on commercial software, even though both solutions most probably require services, such as maintenance and support. This is often measured by the total cost of ownership (TCO) and as other things might be equal the software solution with the lowest TCO is usually the most financially desirable one.[56] Understanding the TCO is a prerequisite of correctly estimating the return on investment for any kind of software.[57]

Consequently, many public sector players have chosen to use only open source software in their IT systems. The city of Munich switched all of their desktop computers to run a

[53] Ballardini 2012 p. 8.
[54] Ballardini 2012 p. 8.
[55] Stallman 1999 p. 53.
[56] http://en.wikipedia.org/wiki/Total_cost_of_ownership, last visited 22.4.2014.
[57] Deek-McHugh 2008 p. 270.

customized version of Linux ("LiMux"), mainly for being less dependent on large software companies but as well as for budgetary reasons.[58] As a contrast, vendors of proprietary software often claim that although initial costs may be lower for open source software, their higher long-term costs make them economically less attractive for companies.[59] The benefits of open source software can be divided into short-term and long-term benefits.[60]

Short-term benefits:

- Low (possibly zero) purchase price
- No need to pay for the amount of copies (licenses) in use
- Reduced need for regular upgrades (resulting in lower upgrade fees and lower management costs)
- Longer uptimes and reduced need for systems administrators
- Near-zero vulnerability to viruses eliminating need for virus frequent checking, data loss and downtime
- Claimed lower vulnerability to security breaches and hack attacks reducing systems administration load
- Claimed ability to prolong life of older hardware while retaining performance

Long-term benefits:

- Better adherence to standards permits competition in the market, reducing vendor lock-in and consequent monopoly pricing
- Availability of source code provides greater continuity and security against
    - Financial collapse of vendors of key products
    - Vendors choosing to withdraw support for unprofitable products
- Protection against being required to upgrade to vendor's newer products

It should be noted that the above merely present some of the most commonly listed benefits of using open source and as open source software vary, no clear-cut conclusions should be made from the above list.

In addition to the above list, the most cost-effective factor of open source software probably relates back to openness as it is the core value and leading principle, as openness draws in feedback and patches from developers around the world. Evolution of the software has become streamlined since every iteration is based on the needs of a critical mass of users.

---

[58] "We did a calculation and we made it publicly available on our information system for the city council. We have the exact same parameters for staying with Windows as with the migration to the Linux platform. Based on those parameters, Linux has saved us €10m." Available at: http://opensource.com/government/14/5/how-munich-switched-15000-pcs-windows-linux?, last visited 20.5.2014.

[59] Deek-McHugh 2008 p. 266.

[60] The above division and list are based on the UK Government Study. Peeling & Satchell, QinetiQ, 2001.

Furthermore, as those iterations are shared on a source code level, every developer in a company using the particular piece of open source software won't have to reinvent the wheel every time.[61] In this sense, the iterations and collaboration of masses resemble the principles of open innovation where the boundaries on innovation is not limited to companies.[62]

Flexibility and Freedom

When it comes to *flexibility and freedom*, the most challenging part related to commercial software is to avoid vendor lock-in. In this context, lock-in means that a company may not be able to use or implement the additional solutions it desires or deems most suitable for a particular purpose, but rather has to revert to an inferior and often more expensive solution which is more often than not offered by the same vendor as the main software solution.

Lock-ins usually result in IT systems dictating business needs while it is obvious that the state of affairs should be completely the opposite. As requirements change in business, the chosen IT solutions should silently adhere with the revised requirements or they should be easily swapped out for solutions that fulfill such requirements. As companies undoubtedly have various IT systems, they should be able to be interlinked and interoperable in a manner which facilitates and supports the company's business purposes. For this purpose, openness of open source software allows the developing community as well as corporate members to build open interfaces which are based on universal standards and formats in order to facilitate compatibility.

As opposed to open standards, vendor lock-in is most commonplace when proprietary data formats are used. According to Dolmans, once an industry wide standard has been agreed, the industry becomes locked in, hence "an opportunity for unscrupulous IPR owners whose intellectual property is essential to the standard".[63] Striving for compatibility in these types of scenarios might in practice become such a big nuisance, as data conversion tools and possible application programming interfaces are not available as they might be undocumented or even protected (by IPRs, contract and

[61] Further on the argumentation, see e.g. Rowe 2008.
[62] See e.g. Chesbrough 2003. For the concept of open innovation: http://en.wikipedia.org/wiki/Open_innovation, last visited 22.4.2014.
[63] Dolmans 2010 p. 119-120. Dolmans goes on by claiming that some standard setting bodies are too influenced by industry leaders. As an example, Dolmans mentions Microsoft's suspicious acts when trying to adopt OOXML as the new standard for presenting spreadsheets, charts, presentations and word processing documents.

technical means), that it becomes less complicated to just “choose” the original vendor’s solution. As this process is repeated, even a large software company might be subjected to vendor lock-in without even noticing it.

According to Deek-Mc Hugh, vendor lock-in is unsurprisingly most commonplace with proprietary software and systems.[64] Open source projects do not have any commonly known motivation to attempt vendor lock-in, although companies providing support for open source software might. Since there are no business related means to justify other than the use of open standards and formats, adherence to state-of-the-art standards is typically high. Furthermore, if sub-par or obsolete formats were to be used, the principle of sharing the source code results in someone in the community releasing a version that is the most compatible and hence most usable. This is probably one of the reasons why open source software is so popular in infrastructure areas. As an example, the Apache Server software is more popular than its commercial rivals, probably due to its strong reliance on the common http(s)-protocol.[65]

The drawback for using standards and common formats is that they lag behind in terms of individual and additional features. The ability to sell these products depend on the vigilance of their sales force and marketing, but also market research in order to capture and thus monopolize areas not covered by open source. At least for infrastructure solutions, open source software might be the solution for mitigating the risk of getting locked-in, and, furthermore, adhering to common standards.[66] Ultimately spending resources on open standards and interoperability pays off as it provides for the company’s IT infrastructure to be lean and modular in order to better adapt to changing business requirements. As an example, Netflix has incorporated open source as a core item in their strategy and have open sourced most of its cloud software tools to other

[64] Deek-McHugh 2008 p. 267.
[65]Further argumentation: “Any vendor that controlled the lions' share of the browser *and* the server market would feel strongly tempted to exclude competitors by proprietary extensions to the HTTP protocol if they thought they could get away with it. No single vendor has yet managed to control both ends of this equation to a great enough degree.” [65] Peeling & Satchell, QinetiQ, 2001.
[66] "It is the use of proprietary standards and protocols that effectively mandates the purchase of further products from the same supplier. Mandating the use of open internet standards (as in the e-GIF) rather than proprietary formats, and developing XML-based data definitions, for intra-Government, and Government-to-Citizen interoperability, is a practical approach to controlling the [proprietary lock-in]". Peeling & Satchell, QinetiQ, 2001.

cloud services providers. The reasoning behind this is to be able to procure scalable cloud services which suit its needs from different service providers.[67]

Even if a company's resources are limited, using open source components might prove to be cost saving from another aspect as well. Often software needs to be bespoke to better suit and address particular needs. However, bespoke software and providing support is more often than not a question of resources. As most open source projects are led and receive contributions from the developing community, even burdensome and complicated modification may be achieved through the community. Furthermore, as the community grows, it might prove that a company's needs for software tailoring might not be so unique after all as other companies may have similar needs. Hence a demand for such services exists and will most likely be addressed from within the open source community.

Support and Accountability

A common argument from commercial software vendors is that they provide product warranties and are ultimately liable for their products and services. This argument is only valid when it comes to claims relating to IPR infringement where the vendor has an actual economic interest in resolving the issue in the speediest manner possible as IPRs are often the core asset of a software product and their loss would render the product or service at obsolete in its substantial parts.

However, anyone who has ever been involved in software procurement or even read a license agreement can rather quickly ascertain that inaccurateness of the above argument as license agreements in large disclaim all warranties to the fullest extent allowed by applicable laws and restrict and limit their liability in incident cases. The purpose of these agreements, which are often accepted in the form of click-wrap or shrink-wrap agreements, is to disclaim and limit the liability insofar as it is allowed by

[67] It is known that currently Netflix relies heavily on Amazon Web Services, and it only seems logical that by offering its cloud tools to the whole industry, it might bolster competition among the service providers and Netflix will ultimately be less dependent on one or a few service providers. http://osdelivers.blackducksoftware.com/2014/01/22/top-ten-foss-legal-developments-in-2013/, last visited 1.10.2014.

law.[68] In other words, the license agreements are in place to abstain the vendor from any liability rather than to direct liability to the appropriate party.[69]

Support exists for open source software in the same way it exists for commercial software. Indeed, most commercial software do not include support either. However, the arguments in favour of using open source in regards to support are basically the same ones that have been repeatedly brought up in the previous subchapters. As the source code is available, the support provided isn't connected to a certain company, quite the opposite, as anyone having the skills may offer support to open source components. More importantly, the support isn't contractually bound to e.g. the initial author or vendor of the software.

Moreover, for the most scrutinized and biggest open source projects, support might not even become a tangible issue in the first place as the reliability of the software might in practice be on such a high level that the need for support diminishes or even ceases completely.[70]

Efficiency Perspective

According to Ballardini, one major advantage from a company's point of view in wide adoption of open source software is the recruiting process.[71] Incorporating most common and well known open source projects in the company's products and IT processes may boost its effectiveness on the labour market as it is able to recruit software developers already fully familiar with its software architecture.[72] This might well have its advantages and may result in more efficient use of resources through speedier product development.

2.3.2 Open Source Business Models

In this subchapter, the main forms of open source business models are discussed briefly. The object is to merely point out what kind of economic and business related motives

---

[68] On the acceptance method and typical conditions in click-wrap or shrink-wrap license agreements, see e.g. Välimäki 2009 p. 157-158.
[69] See Ballardini 2012 p. 35-36 in particular how clients of software companies view the risk for the lack of e.g. warranties and indemnification equal in regards to commercial software and open source software.
[70] "Speaking from a personal perspective, the authors feel obliged to point out that when software is as reliable as Apache, MySQL and similar projects, support dwindles as an issue. When software works reliably, support for it ceases to be a frontline concern". Peeling & Satchell, QinetiQ, 2001.
[71] Ballardini 2012 p. 9.
[72] Ballardini 2012 p. 9-10.

software companies have in relation to open source and how they are monetizing them in practice.

Some authors have categorized software business models into three main categories, often comprising of companies offering traditional commercial software and services, companies offering open source software and services and finally, and most interestingly, companies offering both commercial and open source products and services.[73] An increasing number of software companies rely on both commercial and open source software and some, like Microsoft, draw a clear distinction between their commercial and open source products, whereas others, like IBM, offer truly hybrid products and services.[74]

However, traditional commercial software companies have begun to comprehend the business logic of using open source, if not completely, but rather as a supplemental alternative. For instance, Microsoft offers cloud services on different platforms, including open source. This has probably more to do with growing demand for open source solutions rather than Microsoft's business model paradigm shift, but it clearly illustrates that the gap and distinction between commercial and open source software has narrowed as software is offered as a service.[75]

The below list merely consists of examples of business models using open source software. For this purpose, an attempt to categorize a business constantly on the move might be somewhat futile as it is not the main subject of this thesis and would more than likely merely result in an unsatisfactory compromise.

For the purposes of this thesis, the main open source business models are divided into three main categories, as described in more detail below.

Dual licensing refers to a licensing model where a software company offers its software simultaneously under open source and commercial licenses.[76] The business logic for a dual licensing model varies from anything between financing the open source project through the offering of a commercial version for a charge and offering a stripped down

---

[73] On the categorization, in particular in relation to hybrid business models, see e.g. Ballardini 2012 p.11.
[74] http://www.ibm.com/developerworks/opensource/newto, last visited 23.4.2014.
[75] On Microsoft offering open source solutions for cloud computing: http://msopentech.com/blog/2014/01/27/ms-open-tech-releases-open-source-microsoft-cloud-server-chassis-manager-software/, last visited 23.4.2014.
[76] More on the concept of dual licensing, see e.g. Meeker 2008 p. 278 and Välimäki 2003, available at http://www.valimaki.com/org/dual_licensing.pdf, last visited 24.4.2014.

open source version in order to upsell the commercial version to the customer. The dual licensing model requires that the company or entity offering the commercial version has the ownership of all relevant IPRs related to the software. Therefore it can control the development of the core product and is able to license it with different licenses.

In some business models the open source and commercial versions are identical, whereas in the likely more common scenario the commercial version comes with extensive features and support.[77] However, the most significant difference is the terms under which the software is licensed where the open source version is usually licensed under e.g. GPL v.2 or v.3, the commercial version is relieved from copyleft and sharing obligations usually connected to such licenses. Furthermore, it is common for the commercial version to have more extensive warranties and support obligations (e.g. in form of a service level agreement).

According to Välimäki, concerns regarding copyleft obligations are often the reason for companies to acquire commercial licenses which might directly be linked to the fact that dual licensing companies often, if not always, offer the open source version through a strong copyleft license in order to steer the potential paying clientele towards the commercial version.[78] An often used example of a software company relying on the dual licensing model is MySQL which offers its database solutions under GPL v.2 as well as a commercial license.

Offering professional open source services is a valid business model used by, in particular, various smaller software companies. However, also large companies like IBM and RedHat have started to rely heavily on this business model as well, although large companies may have multiple business models for their products. Professional services may include e.g. training, technical support and consulting. In this business model the earnings logic is that although the open source software might be free of charge for the customer it may still require modifications, bespoke functionalities and support which the company offering open source services may bill as a project or even by the hour.

As the trend seems to be heading towards software companies offering services and consultation rather than static products and licenses it is rather logical that even larger software companies do no longer restrain themselves only to commercial software.

---

[77] Meeker calls the former licensing model the pure dual licensing model. Meeker 2008 p. 278.
[78] Välimäki 2003 p. 16 (in PDF version).

Offering software as a service seems to be the biggest trend in the whole software industry during the last years and it only seems to accelerate, as has been noted in the previous chapters. The business model fits in well with open source as the client operates the service from the cloud, for instance, through a web browser. In this context, the license under which the underlying software is licensed doesn't carry as much significance as opposed to if it was installed on the client's hardware. Furthermore, as providing software as a service through the cloud, most of the copyleft and other sharing obligations don't apply, although it has to be noted that there are exceptions even to this.[79]

From a real life point of view, it should be noted that almost all companies use open source software in some way whether it is only for internal purposes, such as internal processes or part of product development. For instance, the majority of tools needed in order to develop mobile applications for Google's Android mobile operating system are open source.[80] Furthermore, a large portion of commonly used plugins, libraries and other software supporting software development or offering standard solutions to common problems are, if not completely, at least available as open source versions.[81]

[79] Most notable exception is the Affero General Public License (AGPL) for which copyleft obligations apply even if the software is distributed through a network. See Section 13 of the AGPL, available at: http://www.gnu.org/licenses/agpl-3.0.html, last visited 24.4.2014.
"Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph."
[80] For more information, see http://source.android.com/accessories/index.html, last visited 28.4.2014.
[81] As an example, most commonly used database solutions are available as open source versions. Five of the most used database solutions out of ten are licensed under open source licenses, see http://db-engines.com/en/ranking, last visited 28.4.2014.

3. Legal Risks of Using Open Source Software in Companies

As will be discussed in this chapter, open source licenses are enforceable and come with numerous obligations with which the licensees have to comply with. While some of these obligations are quite straightforward and do not present any problem from a compliance point of view, some obligations and their legal consequences are far more obscure and hence present tangible risks to software companies.

In this chapter the most common risks are identified and categorized. Furthermore, the legal implications of non-compliance is analyzed. The chapter starts with an introduction on general principles of copyright protection for software and its economic justification and is followed by how different open source licenses have been enforced in different jurisdictions around the world. The chapter then continues with providing some input on what are the major obstacles for enforcing open source licenses. Furthermore, the chapter presents some of the most common terms in open source license issues for legal compliance regarding such key concepts as copyleft, derivative works and distribution, which are presented from a general copyright law perspective and then that perspective is applied to open source licenses in order to provide a more practical perspective from a company's point of view. Although copyleft is not a legal term, it has legal implications which depend on how it is implemented in specific open source licenses. Therefore it is studied together and in the same context as derivative works and distribution as these concepts are closely linked. In the context of copyleft, only the GPL (versions 2 and 3) as well as the AGPL will be studied as they have been identified as the most controversial open source licenses from a legal point of view.

3.1 Copyright as the Legal Protection for Software

In the EU the protection of software is based on the Software Copyright Directive[82], which was approved in 1991. The aim of the directive was to harmonize copyright law concerning software in the EU. In Finland the special provisions for software were incorporated as part of the Finnish Copyright Act (CRA)[83] the same year. Since then the

[82] Council Directive 91/250/EEC of May 1991 on the legal protection of computer programs.
[83] Finnish Copyright Act, 8.7.1961/404.

provisions have been amended several times. The directive filled a known gap in European legislation and harmonized copyright protection for software in the EU.[84]

According to Article 1 of the CRA, software is protected by copyright as a literary work. In order to acquire copyright protection, software has to meet the "work threshold", meaning that the software has to be both independent and original, in the sense that it is the author's own and independent intellectual creation.[85] The work threshold for software is fairly low.[86] According to the CRA's legislative history, creativity and originality manifest themselves primarily in choices made by the author when executing a solution for a data processing problem. If there is only one possible solution to the problem in question and the solution is the result of purely mechanical requirements, the software does not represent the author's creative and original contribution and is hence not eligible for copyright protection.[87]

According to Article 1 of the Software Copyright Directive, the protection in accordance with the directive applies only to the expression of a computer program. The ideas and principles which underlie any element of a computer program, including those, which underlie its interfaces, are not protected by copyright. This choice of copyright protection has been widely criticized in legal literature. For instance, Ballardini considers that the shortcomings of legal protection for software has lead to the adoption of patent protection for software related inventions.[88]

The status of the right holder of copyright to software is conferred to the author of the software, that is, to the programmer who writes the source code. If more people are involved in the coding process, which often is the case particularly in companies, software is considered to be a joint work (Article 6 of the CRA) or a compiled work (Article 5 of the CRA).[89] Where software is created by an employee in the course of his duties, or following the instructions given by the employer, the copyright to the software is automatically transferred to the employer pursuant to Article 40b of the CRA.

---

[84] Harmonization was well needed as standards for protection varied widely. See Software Directive, recital 1.
[85] Article 1. Paragraph 3 of the Software Copyright Directive.
[86] Välimäki 2009 p. 19.
[87] Government Bill, HE 161/1990, preamble of 1 §.
[88] Ballardini 2012 p. 7.
[89] Välimäki 2009 p. 26–30.

Copyright renders the right holder both moral and economic rights to the work. Moral rights protect the personal and reputational value of the author and the work and are not in principal transferrable. Moral rights include, among others, the right to be acknowledged as the author and the right to object derogatory treatment of the work (Article 3 of the CRA). Economic rights provide the author the exclusive right to control the work by making copies thereof and by making it available to the public whether in unaltered or altered form, in translation or adaptation, in another literary or artistic form, or in any other forms created by using other technical means. This in turn means that no one else is permitted to make copies of the software nor make it available to the public without the author's consent. As running software includes copying of the work as protected by copyright law, the customer or end user of the software needs a license (i.e. a permission to use) from the author or right holder in order not to infringe the author's exclusive rights.[90]

The term of copyright protection lasts for the lifetime of the author and for seventy years following his death or, in case of a joint or compiled work, after the death of the last surviving co-author (Article 43 of the CRA). After this time period, software becomes part of the public domain, meaning that it can be copied and distributed without the author's consent or payment of license fees.

As this subchapter sets the basis for copyright protection for software, it should be noted that also other forms of IP protection are available for software. For instance and even though copyright protects the source code of software, software companies do not disclose their source code but rather protect it as a trade secret. This is often accompanied and enforced through contractual obligations set out in a license agreement. Furthermore, patent and design protection might be available for the same or nearly the same subject matter as for copyrightable subject matter.

3.2 Scope of Protection

The appropriate form of protection for software has been subject to numerous legal, economic and sociological discussions. Even though copyright is an old form of IP protection and is not designed with particularly software in mind, it strives to balance between an appropriate level of protection through predefined monopoly rights and

[90] Bainbridge 2008 p. 53.

providing incentives for software innovation.[91] Copyright as means of protection for software is an optimal policy if the rents gained from the monopoly provided by copyright protection are in balance with the total welfare it produces to society.[92]

Both from an economics as well from a practical point of view copyright was chosen as means of protection for software since protecting software as literary works, in spite of their technical nature and attributes, means that protection is only granted to expression, not the underlying idea. Consequently, copyright protection does not prevent competitors from expressing the very same idea in a different way. This dichotomy of idea and expression is important in order to prevent monopoly rights and appropriating undeserved value for the author.[93] Although the distinction between expression and idea is essential to copyright law, it has proven to be rather difficult a task for the courts to determine which parts of a program are, in fact, expressions of a certain idea.[94] It has been confirmed by the European Court of Justice ("ECJ") that functionality of software is outside the scope of protection. The ECJ rules in the case of SAS Institute that non-literal copying of software, i.e. without reference to the underlying source code, is not considered as copyright infringement.[95]

Because of the fundamental limitation in the scope of protection, algorithms, program libraries, main technical principles or other solutions of pure technical nature of software are generally not protected by copyright law.[96] Hence, at least in theory, these kind of technical solutions become available to competitors as the software is published.[97] However, this is not the case in practice since software companies do not generally disclose their commercial software on a source code level and also tend to limit the right to inspect and reverse engineer the software to a maximum extent allowed by applicable law.[98]

As copyright protects expression (i.e. first and foremost the source code) which in turn means that the most time-consuming and expensive parts in creating software, such as the development of the structure and logic of the software, are left outside the scope of

[91] Välimäki 2009 p. 13.
[92] Christie 1994 p. 487.
[93] Christie 1994 p. 488.
[94] Christie 1994 p. 493.
[95] SAS Institute Inc. v World Programming Ltd, C-406/10, 2 May 2012.
[96] See e.g. Bainbridge 2008 p. 67-74.
[97] Välimäki 2009 p. 15.
[98] The Software Copyright Directive provides for a limited right to reverse engineer software in e.g. for purposes of obtaining interoperability information if it's not provided by the software vendor.

copyright protection.[99] Consequently, it can be concluded that copyright protection actually leaves out a majority of the total development costs. This means that the author is unlikely to fully recover his development costs, at least if relying on copyright protection alone, which in turn might result in a disincentive for creation of new software.[100] In contrast, since copyright protection doesn't cover all aspects of software and some aspects regarding the scope of protection remains uncertain (such as APIs or programming languages[101]), authors of software spend excessive resources on decryption and other means to conceal the origin of the source code in order to prevent or slow down competitors from direct copying. This, in turn, might stifle innovation as the greater the market potential is, the greater the need for adequate IPR protection.[102] In this sense open source licenses offer a remedy in order for copyright protection to yield the benefits (i.e. disclosure of source code) which it was initially meant to from an economics point of view.

The above scope of copyright protection for software means that as a starting point copyright protection does not actually hinder the making of competing or substitute software products, except when identical products are made by literal copying of the source code.[103] Bainbridge claims that even a small modification might require a significant amount of investment, labour and skill and yet it might not amount to a level which might acquire copyright protection.[104] Therefore it can be argued that copyright protection for software is not actually an exclusive right at all.

Copyright protection for software has been under constant criticism since its adoption. Many alternatives are presented, foremost sui generis and patent protection. Also other forms of protection which some might consider exotic or far-fetched, have been introduced.[105]

3.3 Open Source Licenses as Enforceable Agreements

As the software industry has been largely US-centric, it is perfectly sensible that most of the litigation regarding enforcement of open source licenses come from the US as it is

[99] Brown 1987 p. 6.
[100] Samuelson 2003-2004 p. 16 and Välimäki 2009 p. 26.
[101] See e.g. Bainbridge 2008 p. 70-73.
[102] Landes – Posner 1989 p. 330.
[103] Välimäki 2009 p. 15-16.
[104] Bainbridge 2008 p. 66.
[105] Christie suggests some sort of design right since he argues that the most valuable part of a computer program is its design. See Christie 1994 p. 493.

the home market for many major software companies. US-centricity of litigation might also relate to the nature of the US legal system as it offers possibilities to "try and fail" since the plaintiff is not subject to such a strong financial burden regarding to the obligation to reimburse the respondent for its legal fees as in most European jurisdictions.[106]

Another reason for the differences might be the fragmented legal framework even inside the European Union as every member state has its own legislation regarding intellectual property and contract law. Hence court judgments on similar matters might differ from country to country. As copyright legislation in the EU has been harmonized through directives rather (as opposed to EU Regulations), every member state may choose the precise contents and methods how they implement the directives into their national legislation.[107] Consequently, this fragmentation might lead to different interpretations of similar issues between the member states. Since European case law on open source is scarce, it isn't sensible to speculate on the matter at this point.[108]

Firstly, it should be noted that the open source license obligations have been written the way they are for specific reasons. Even though most of these are originally based on fundamental reasons regarding how software should be available without cost for everyone, it should be understood that the motivation for e.g. the strong copyleft obligations can be considered as enforcement mechanisms in order to ensure that software companies are not able to benefit from the IPRs included in open source software without doing their part (i.e. complying with the license obligations).[109]

In this sense, the copyleft obligations enforce the licensee to comply with the license by e.g. releasing the source code to modifications, as opposed to just reaping the benefits of an open source solution through making a few enhancements and then offering the whole package as its own product, for a fee.[110]

Secondly, it should be borne in mind that even though litigation regarding open source licenses exists, the scarcity of actual court cases is more likely intentional rather than

---

[106] Contrary to the rule prevalent in most European countries, the US legal system does not generally oblige the losing party to reimburse the other party for its legal fees. For more, see e.g. Derfner, Mary F. – Wolf, Arthur D, "Court Awarded Attorney's Fees", 1992.
[107] Tritton et al 2008 p. 16.
[108] For more information on European case law regarding open source litigation, see e.g. Van den Brande et al 2014 where case law has been categorized by country.
[109] Evans et al 2008 p. 76-77.
[110] Carver 2005 p. 453.

implicating that the actual number open source license infringements is low or non-existent. This poses some further questions as to why this might be.

On the one hand major software companies are stakeholders in the largest open source projects, such as the Linux Foundation and the Apache Foundation. Even though legal scholars and corporate legal counsels are eager to find out the legal boundaries of e.g. copyleft obligations, it's not in their best interest to stir the hornet's nest as the most potential infringers are likely to be partners of the same foundations. In this context, only the most blatant infringements would be likely to result in actual court action. The companies involved in open source are more than likely fully aware of this.

On the other hand, the open source community and the software companies involved want that open source projects keep running and keep receiving funding from the industry. For this purpose, as not knowing the clear boundaries of certain open source license obligations is the current state of things, the community seems somewhat content with continuing conducting their business as usual rather than engaging in lengthy legal disputes in order to gain legal certainty. As has been presented in earlier chapters, even most major software companies rely heavily on open source software and are, therefore, reluctant to initiate anything that might jeopardize their current business models. In other words, the benefits of conducting business outweigh the benefits of legal certainty.

Thirdly, as another theory for the low number of court cases regarding enforcement of open source licenses, litigation might be scarce as the copyright holders are the only ones who can rightfully initiate court proceedings.[111] Even more so, the identification of the right holder might be challenging in an open source project as the project may have been forked in many directions over time even though copyrights are often assigned to the foundations managing the projects.[112]

Any intermediate distributors may only enforce the license obligations insofar as they clearly and solely relate to the modifications they might have made to the original open source distribution.[113] However, it seems that litigation is restricted to cases where copyright ownership is clear and the nature of the alleged infringement or license breach is vital to uphold the credibility of open source license obligations. In this sense,

[111] Meeker 2008 p. 170-171.
[112] On open source forking, see e.g. Evans et al 2008 p. 91 (Linux forking) and on software forking in general http://en.wikipedia.org/wiki/Fork_%28software_development%29, last visited 6.5.2014.
[113] Meeker 2008 p. 172.

enforcement cases work as a reminder for companies who are attempting to try the boundaries of the licenses. Even though a company might end up facing legal charges in court, the process may well provide valuable information on how far it can go before facing legal action.

In fact, one of the most influential open source community advocates, the Free Software Foundation, has only enforced its open source licenses in court on a handful of occasions. According to Eben Moglen, the general counsel for the Free Software Foundation, open source litigation is feared across the industry so cases almost never end up in court.[114] Hietanen stresses the fact that as open source licenses have simply codified previously unwritten rules of the open source community, the need for enforcement has been low.[115] It might even be that as a great number of companies using open source software in their business are unaware of their non-compliance with license obligations, they correct their behavior without undue delay.[116] Another factor is that the Free Software Foundation has, in case non-compliance is not remedied by the company developing the software, contacted the vendor's customers directly by informing them that the vendor is not complying with its license obligations.[117] Such customers have then presumably applied pressure to non-compliant vendors by threatening to switch to competing products.[118]

It has been established that open source licenses are enforceable both in Europe and the US. Consequently, some relevant case law will be presented in this subchapter. As Europe is not a single jurisdiction, only some of the bigger jurisdictions are dealt with here, with the exception of Finland.

Although no case law exists in Finland, it is clear that open source licenses are enforceable under Finnish law as the law does not have any prerequisites for granting copyright licenses.[119] It is up to the right holder to decide whether he will want to enforce the rights as copyright infringement or breach of contract.[120]

---

[114] http://moglen.law.columbia.edu/publications/lu-13.html, last visited 13.5.2014.
[115] Hietanen 2008 p. 45.
[116] Ferguson 2006 p. 406.
[117] Ferguson 2006 p. 407.
[118] Ferguson 2006 p. 407.
[119] Von Willebrand – Tanskanen in Van den Brande et al 2014 p. 82-83.
[120] Von Willebrand – Tanskanen in Van den Brande et al 2014 p. 83.

In France, the relationship between the licensor and licensee is regarded as a contractual one as anyone accepting conditions of a license enters into a contractual relationship with the right holder.[121] However, the enforceability of open source licenses have been tested in court as well. In a decision issued by the Paris Court of Appeal in 2009, the court concluded that the respondent had failed to comply with the terms of the GPL v.2 as it had included in its delivery a modified version of an open source component without complying with the copyleft obligations (i.e. providing source code for the modification) and the obligation to deliver the full license text along with the software.[122] The case has received controversial legal commentary as e.g. Perbost and Walter regard the case as significant since before the court decision it was feared that the obligations in open source licenses might lack legal effect.[123] Von Willebrand, however, regards the implications of the case far less important as, although the court did decide that license terms of the GPL are enforceable, its terms played only a limited role in the decision as the main content of the dispute was whether the vendor had delivered software in accordance with the delivery agreement.[124]

In Germany open source licenses are commonly seen as license agreements which are regarded as contracts.[125] Therefore the validity of the license should be construed by assessing whether a contract has been entered into by offer and acceptance.[126] This same principle can be applied to most European countries. In a German court case the Munich District Court ruled for a preliminary injunction against UK company Fortinet for its alleged non-compliance with the GPL.[127] In addition, the plaintiff claimed that Fortinet was obfuscating the existence of GPL source code in its product. The case was settled when Fortinet agreed to make changes to its product by making available certain parts of the source code as required by the GPL.

In a more recent case, the Regional Court of Hamburg ruled that Fantec GmbH, a manufacturer of media players and streaming services, had violated version 2 of the GPL

---

[121] Perbost – Walter in Van den Brande et al 2014 p. 102-103.
[122] Cour d'Appel de Paris, Pôle 5, Chambre 10, no: 294, issued on 16 September 2009, available at http://fsffrance.org/news/arret-ca-paris-16.09.2009.pdf, last visited 29.10.2014.
[123] Perbost – Walter in Van den Brande et al 2014 p. 107.
[124] Von Willebrand 2009 p. 3.
[125] Engelhardt – Jaeger in Van den Brande et al 2014 p. 128-129.
[126] Engelhardt – Jaeger in Van den Brande et al 2014 p. 129.
[127] District Court of Munich I, 21 O 6123/04, May 19, 2004, In re Welte v Fortinet UK Ltd(text in English available at http://www.oii.ox.ac.uk/resources/feedback/OIIFB_GPL2_20040903.pdf, last visited 28.10.2014. For commentary see e.g. Westkamp 2008.

by not shipping the source code of the modified open source component.[128] The court decided that Fantec had acted negligently by not complying with the obligation to provide source code upon distribution and ordered Fantec to comply with the copyleft obligations subject to a penalty payment.

In the United Kingdom no court cases are known to exist. However, enforcement should be available if the general conditions on contract validity in English and Welsh law are met. These include explicit acceptance of the license terms.[129] The method of acceptance per se is not the defining factor, so it has been concluded that acceptance might be validly given even through click-wrap or shrink-wrap.[130]

One of Free Software Foundation's better known cases in the US is the case brought against Cisco, one of the largest producers of network hardware and services, for its alleged breach of its copyleft obligations by failing to offer the source code to the modifications it had incorporated into its products.[131] The case was settled shortly after Cisco agreed to appoint an open source compliance officer to monitor proper license compliance in its products.[132]

Apart from litigation in courts, open source community advocates, such as the Free Software Foundation or Harald Welte's GPL Violations website, use other means of policing open source (mainly GPL related) software license violations.[133] For instance, the Free Software Foundation have filed numerous amicus briefs in the US in order to sway the courts in their favor.[134] From a research point of view, this is a shame since they might not always be public. However, their legal relevance can be questioned. Furthermore, most of the cases involving open source litigation rather end up in settlement than in a full scale court judgment, let alone in a court judgment of such stature that would have actual value as a legal precedent.

---

[128] Landgericht Hamburg Az: 308 O 10/13, 14.6.2013. Available in German at http://www.ifross.org/sites/default/files/130618%20Urteil%20Fantec.pdf, last visited 6.5.2014. See also http://www.gpl-violations.org/news/20130626-fantec_judgement.html, last visited 6.5.2014.
[129] Katz – Mitchell in Van den Brande et al 2014 p. 443-444.
[130] Meaning clicking "Accept" or "OK" in a computer interface, as applicable. More in Katz – Mitchell in Van den Brande et al 2014 p. 443.
[131] http://www.fsf.org/news/2008-12-cisco-suit, last visited 6.5.2014.
[132] http://arstechnica.com/information-technology/2009/05/cisco-settles-fsf-gpl-lawsuit-appoints-compliance-officer/ and https://www.fsf.org/news/2009-05-cisco-settlement.html, last visited 6.5.2014.
[133] See http://www.gpl-violations.org/about.html#why, last visited 6.5.2014.
[134] Meeker 2008 p. 172.

An issue worth mentioning, although not analyzed in this thesis, is that many US legal scholars have contemplated whether an open source license, such as the GPL, is a valid agreement or merely a license.[135] The framing of the this question might have different implications as to whether a valid agreement has been concluded in the first place regarding consideration which is a prerequisite in US contract law for a binding contract.[136]

Furthermore, the remedies applicable would differ greatly as a contract would be enforceable to the extent that a non-compliant party could be forced by the court to release the source code of a derivative work whereas as a license would merely compel the court to order the licensee to cease use of license.[137] In addition, licenses are under federal copyright legislation and therefore harmonized across the US, and even to some extent by obligations from international treaties such as the WTO Agreement on Trade-Related Aspects of Intellectual Property Rights from 1994, whereas contracts are under state legislation and thus subject to variation between different states.[138]

Consequently, it is unsurprising that proponents of open source software as well as software companies have claimed that open source licenses are, in fact, licenses rather than contracts as the remedies available for licenses would be, in a worst case scenario, less devastating for licensees than if open source licenses were to be considered as contracts. For instance, the Free Software Foundation has officially and explicitly advocated that the GPL is a license.[139]

Although the aforementioned is the current status, we may only know for certain once e.g. an appellate court takes stance on the matter. In fact, Meeker challenges the views of most open source proponents by calling their arguments as “inaccurate statements of law”.[140] Her key arguments include that most countries do not make the same distinction between license and contract as it is known in the US.

Furthermore, Meeker goes on by stating that the question should not be whether a specific open source license is in fact a license or a contract, but rather how the license

---

[135] On the juxtaposition between license and contract, see e.g. Gonzalez 2007 p. 187 and Meeker 2008 p. 223-232.
[136] Gonzalez 2007 p. 187.
[137] Moglen 2001, available at http://www.gnu.org/philosophy/enforcing-gpl.html, and Jones 2003 and Jones 2003, available at https://lwn.net/Articles/61292/, last visited 6.5.2014.
[138] Jones 2003.
[139] Gonzalez 2007 p. 188 and footnote 188.
[140] Meeker 2008 p. 225.

has been implemented, i.e. whether the user has actually accepted the terms prior to using the applicable software.[141] Another often repeated argument is that open source licenses have not been properly tested in courts so they might not be enforceable. Meeker addresses this by stating that licenses or agreements do not need to be tested or approved by courts for them to be valid, i.e. adjudged or confirmed to be enforceable.[142]

From a European perspective, the distinction between agreement and license is trivial per se, but a similar question deriving from this line of thinking is whether a contractual relationship has actually been established, especially in cases where the open source license isn't visible and readable to the licensee, as the case often is. According to van Holst, most common open source obligations are likely to be treated as "negative obligations of the licensee, which automatically make the license a bilateral contract".[143]

Furthermore, an open source "package" might include dozens of other components, all licensed under different open source and free software licenses. A software company might not be successful in claiming that it was unaware of the contents and obligations of the licenses, but for an individual developer this might be a viable claim. Furthermore, as open source components are often part of a commercial product or service offered to consumers, questions regarding what terms (and in what manner) have actually been accepted by the end-user. However, for instance, version 3 of the GPL explicitly states that acceptance of the license is required only when distribution or modifying is involved, not for running or receiving a copy of the software.[144]

From this perspective, it should be acknowledged that all aspects and obligations of all open source licenses might not be similarly enforceable in relation to all users in all jurisdictions. On the contrary, seemingly similar conditions might have implications which differ greatly from each other. As large software companies conduct their business on a

---

[141] Meeker 2008 p. 225-226.
[142] Meeker 2008 p. 176-177.
[143] Van Holst 2013 p. 7.
[144] General Public License, version 3. According to clause 9: "You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so."

global level, it is understandable that these companies are reluctant to enter litigation which might steer the interpretation of open source licenses in an undesirable direction.

To sum up, although litigation involving open source software has been scarce, it should be noted that open source licenses have become an industry-wide and acknowledged tool for conducting software business. A growing number of software companies have recognized the value of open source software and are ready to voluntarily comply with their terms.[145] It could even be that most software companies are, if not diligent and fully aware of all the aspects involved with using open source software in their business, at least attempting to comply with the license obligations. The companies have all the reason to do their best in order to comply with the license terms of open source licenses as the licenses have been declared as valid agreements on almost all major European jurisdictions and the US. Furthermore, some of the key concepts in open source licenses, such as the copyleft obligation, have also been declared as valid.

The question seems to be that in order for open source licenses to be enforceable as valid agreements, it requires explicit acceptance which might differ somewhat from country to country. Moreover, von Willebrand stresses the fact that although open source licenses are valid agreements in most jurisdictions in general, it doesn't automatically mean that each and all elements (i.e. license terms) are enforceable and valid in each particular case.[146] This is however not due to the nature of the license or its conditions but more likely a question of the clarity and possible ambiguity of the license obligation.

3.4 Legal Risks Involved in Open Source Software

Meeker identifies two types of major risks involved with open source software, the infringement risk and the compliance risk.[147] The former means the risk of infringing third party intellectual property by using code generated in collaborative software development and the latter refers to not complying with open source license obligations.[148]

For the purpose of creating legally relevant open source policies, main legal risks should first be identified and analyzed. This way the open source policy becomes a relevant tool

[145] Ferguson 2006 p. 408.
[146] Von Willebrand 2009 p. 3.
[147] Meeker 2008 p. 159.
[148] Meeker 2008 p. 159.

in mitigating common risks and non-compliance with most common (or most relevant) open source licenses for a software company.

For this purpose, the most relevant legal risks involved with the most common licenses are prioritized in this thesis. Therefore the terms *copyleft*, *derivative work* and *distribution* are looked at here in more detail. It should be noted that the abundance of open source licenses carry numerous, on the one hand, very specific obligations which might seem frustrating and challenging to comply with from a software company's perspective. On the other hand, some obligations are very general in nature and hence legally obscure from the licensees' point of view.

3.4.1 Understanding Copyleft

Copyleft refers to the ideology of keeping software free, or at least free of traditional commercial ownership restrictions, by imposing license obligations which ensure that the software is licensed under the same open source license whether it is distributed to third parties or modified.[149] The Free Software Foundation defines copyleft in the following way:

> "*Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. Copyleft guarantees that every user has freedom.*"[150]

The idea behind copyleft is that developers will not be able to reap the benefits of open source software by merely making some modifications and then licensing the whole work under commercial terms. The best known and most widely used strong copyleft licenses are the GPLs, i.e. versions 2 and 3 of the General Public License and the Affero General Public License. According to clause 5 of version 3 of the GPL[151]:

> "*You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:*
>
> - *a) The work must carry prominent notices stating that you modified it, and giving a relevant date.*

[149] Williams 2010 p. 132-133.
[150] Available at http://www.gnu.org/copyleft/copyleft.html, last visited 14.5.2014.
[151] Underlining added in order to highlight relevant parts. Available at: http://www.gnu.org/copyleft/gpl.html, last visited 14.5.2014.

- *b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".*
- *c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.*
- *d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.*

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate."

As presented in the above quotation of the copyleft obligations in version 3 of the GPL, the obligations are numerous and it raises further questions, for instance, what exactly qualifies as "conveying" and when a derivative work is actually considered to be independent from the original work? Moreover, also commonly referred to as the viral question, one of the biggest obstacles for software companies to approving use of strong copyleft licenses relate to the question of when and how, if combined, the entire work should be licensed and thus distributed under a copyleft license.

As opposed to strong copyleft licenses, also weaker copyleft licenses exist. Most commonly named examples of weaker copyleft are the Lesser General Public License ("the LGPL") and the Mozilla Public License ("the MPL"). For instance, the MPL provides that covered source code (code under MPL license) may be mixed and used together with code from other licenses, even commercial ones.[152] As a restriction, the software licensed under the MPL must remain under the same license. Consequently, all derivative works may be licensed on commercial terms thus making it appealing not only

[152] Available at: http://www.mozilla.org/MPL/2.0, last visited 14.5.2014. Underlining added in order to stress relevant parts.

to the open source community but to companies developing commercial software as well.

According to Ballardini, one of the biggest concerns for software companies concerning their use of copyleft licenses have been fear of their commercial software being "forced open" through combining it with open source software.[153] As the previously mentioned represent a great portion of software's companies' legal concerns, this calls for closer examination. As the most used strong copyleft licenses are versions 2 and 3 of the GPL, the below examination will be based on them.

3.4.2 The Derivative Works Question

The derivative works question refers to the concerns raised above which stem from the fear of companies losing control of their valuable intellectual property rights to their own software. Whether this concern is valid or not is discussed in more detail in the next sub-chapter. As Section 5, subsection c of the GPL v.3 states, "you must license the entire work, as a whole, under this License". This is where most software companies and their lawyers hit the breaks as the license doesn't specify what is considered "an entire work". The GPL v.3 merely reads that:

> "A covered work means either the unmodified Program or a work based on the Program."[154]

As the term is not clearly defined, the interpretation of the term should be sought from relevant (national) copyright law, and as copyright law varies somewhat in different jurisdictions, companies may have an unpleasant challenge of assessing its meaning on a global scale. The problem is where the line between an independent work and a derivative work should be drawn as a derivative work must be further licensed under the same reciprocal copyleft license as the original work, whereas a clearly independent work, i.e. a work that is not based on the original, to a certain critical extent, may be licensed free of the said copyleft obligations.[155]

---

[153] Ballardini 2012 p. 23.

[154] Article 0 (Definitions) of the General Public License, version 3.

[155] Jaeger, Till: Presentation on derivative works under European copyright law, at FOSDEM 2013, Legal issues devroom, Brussels, February 2nd, 2013. Slides available at: https://archive.fosdem.org/2013/schedule/event/european_derivative_work/, last visited 20.5.2014.

According to the Free Software Foundation, the term “work based on the Program” should be interpreted as “derivative work” under US copyright law.[156] As “derivative work” is a US legal term, the equivalent term from a European legal perspective would be “adapted work” although it should be noted that the term is not completely equivalent to derivative work.[157] As all relevant legal literature on the matter at hand is from the US and the term derivative work has been highly entrenched in open source vocabulary, that term will be used in this thesis as well.

According to the Finnish copyright act, a work based on an existing work may be protected by copyright.[158] The combination of the original work and the modifications which together result in a derivative work may only be used with both authors’ permission (license). It has been established in legal literature that such adaptation or derivative requires permission from the original copyright holder.[159] From the perspective of the GPL, this permission is conditional on the fact that the licensee licenses the adaptation under the same license as the original.

However, as has been concluded by the ECJ in the SAS Institute decision, right holders may not contractually restrict their licensees from observing and studying software, provided that an copying or reproduction thereof does not go beyond normal use (i.e. loading and running) of software. This decision seems to not have direct implications for open source as open source licenses allow for literal copying of the source code, albeit with certain conditions (i.e. to license to all derivative works under the same license).

As European case law does not currently exist on the concept of adaptation of software, the point of interest in this thesis needs to be directed to legal literature and US case law.

According to Meeker, the question of derivative works is much more complex in the context of software than it is with traditional literary works as a great part of software source code is merely dictated by its function.[160] As it may be, there might well be only

---

[156] For more, see http://www.gnu.org/licenses/gpl-faq.en.html, last visited 28.10.2014.

[157] In this perspective the European copyright law system differs from the US system as European laws do not generally recognize a concept similar to derivative work. E.g. the Finnish Copyright Act states in Article 4 that “If a person, in free connection with a work, has created a new and independent work, his copy right shall not be subject to the right in the original work”.

[158] Article 6, Finnish Copyright Act 8.7.1961/404. Article is based on Article 2 of the Berne Convention

[159] Harenko et al. 2006 p. 57-58.

[160] Meeker 2008 p. 192.

one way to create a certain function in a certain computer language, such as C++ or Java.

Consequently, as copyright law only protects expression and not function, most parts of the source code might not even be protected by copyright in the first place. However, the license might obligate the licensee to a narrower interpretation of copyright law. Much of the source code included in software is, according to Meeker, dictated either by efficiency or by the constraints of the language in which it is written.[161]

As a contrast to the above mentioned, version 2 of the GPL offers a more explicit view on the concept of derivative work. According to Article 2:

> "*You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work … provided that … b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License*."[162]

As the license text states that works that "in whole or in part contains or is derived from", a strict interpretation of the license would suggest that even portions of source code, which might, in terms of copyright law, be insignificant, would trigger the copyleft obligations of the license. However, according to Välimäki, since the license doesn't contain any precise indication of the quantity or quality of the contained source code, it should probably be understood as leaving the matter of interpretation up to applicable copyright law.[163] This would be a practical way of interpreting the license without giving into further legal pedantry.

As copyright law and open source licenses fail to offer a specific definition of what constitutes a derivative work, the point of interest should be directed to legal literature. Välimäki offers a perspective into which aspects may be taken into consideration when assessing the derivative work question by distinguishing three different interpretations:[164]

---

[161] Meeker 2008 p. 190-193.
[162] Underlining and formatting added in order to highlight relevant parts. Available at: http://opensource.org/licenses/GPL-2.0, last visited 20.5.2014.
[163] Välimäki 2005.
[164] Välimäki 2005.

- Source code interpretation
- Component based interpretation
- Communications based interpretation

These three approaches are looked into in further detail.

The first approach, *source code interpretation*, stresses the importance of source code comparison. However, the interpretation method is two-fold as two distinct legal theories, the idea-expression dichotomy and the abstraction-filtration-comparison method, may be considered in order to apply the source code interpretation approach in practice.[165]

The idea-expression dichotomy refers to the very founding principle of copyright law that copyright only protects original expression, i.e. works, and not ideas.[166] Consequently, only expressions of a certain idea, for instance a software company's traffic monitoring application, may be subject to copyright protection whereas the idea itself, i.e. a traffic monitoring application in general, remains without copyright protection. In this context, existing software may be studied and analyzed in order to use them as inspiration for a different interpretation of the same idea. In a software context, according to Välimäki, this would mean that only literal copying of a significant amount of source or object code is restricted.[167] But, in contrast, as pointed out by Meeker, what amount of literal copying of source code is permitted is a complicated question and ultimately defined by courts in individual cases.

The abstraction-filtration-comparison method, also known as the AFC test, refers to US case law (Computer Associates International versus Altai) and has become a dominant way of interpreting derivative works in software cases in the US.[168] Also other tests, such as the analytical dissection test have known to been applied by US courts.[169]

The AFC test includes two phases where the court first abstracts the constituent structural parts of the of the original software and then filters from those structural parts

---

[165] Välimäki 2005.
[166] See e.g. Välimäki 2005.
[167] Välimäki 2005.
[168] Computer Associates International, Inc. v. Altai, Inc., United States Court of Appeals, Second Circuit. June 22, 1992. The test has also been applied in Finnish courts, although indirectly through expert testimony by software professionals, at least in the case of Sonera Systems Oy versus VF Partner Oy (Helsinki Court of Appeals), 28.12.1999.
[169] On the test, see e.g. Meeker 2008 p. 198-199.

all unprotected portions, including, for instance, expression that is necessary or the sole method for accomplishing a certain function and elements taken from the public domain.[170]

As a final part of the test, the court compares any remaining kernels[171] of creative expression to the structure of the comparable software in order to assess whether the software is substantially similar to the original.[172] One might even claim that the AFC test is merely an application of the idea-expression dichotomy in practice even though Välimäki, among others, seem to make a clear distinction between the AFC test and the idea-expression dichotomy.

The second approach, *the component based interpretation*, might be the most suitable for free and open source software as they rely heavily on re-use of source code.[173] This is understandable from the point of view that the very essence of the open source movement was make source code readily and freely available for everyone in order to steer innovation to unchartered areas of software rather than reinventing the wheel due to public unavailability of source code.

Välimäki claims that the component based interpretation ultimately stems from the wording of the GPL which states that the licenses do not apply to identifiable sections that are not derived from the original software and can be reasonably considered as independent and separate works.[174] Companies using strong copyleft licenses have stumbled upon the component based interpretation quite frequently as an often repeated question refers to whether combining open source components with the company's own software triggers the copyleft obligations.

According to Välimäki, the further away the software is from the scope of what is considered copyrightable under copyright law, for instance the case of open source

---

[170] Ravicher 2002: "AFC test for determining whether a computer program is a derivative work of an earlier program was created by the Second Circuit and has since been adopted in the Fifth, Tenth and Eleventh Circuits".
[171] On legal relevance of kernels, see e.g. Meeker 2008 p. 212-215.
[172] Ravicher 2002.
[173] Välimäki 2005.
[174] Välimäki 2005. GPL, version 2, section 2: "If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it."

interfaces used by commercial software versus the case of a commercial component tailored inside open source software, the clearer it should be that such applications should not be considered as derivative works.[175]

The third approach, the communications based interpretation, emphasizes the manner in which software communicates with other software and components and whether this may cause the works to be constituted as derivative. The Free Software Foundation has suggested interpretations which rely both on the means ("mechanisms", according to the Free Software Foundation) of communication such as executables as well as the nature of the communication which defines what kind of information is ultimately interchanged between software.[176]

According to this interpretation, if the modules (i.e. works) are included in the same executable file, they would be considered as a single work whereas if the components were to only communicate through ordinary communications mechanisms such as sockets and command-line arguments, then the works, at the offset, would be considered as separate. As Välimäki points out, copyright does not cover communication mechanisms or protocols per se, and Välimäki, furthermore, seems correct in claiming that the interpretation guide offered by the Free Software Foundation seems closer to patent law rather than copyright law.[177]

Although not covered by Välimäki, legal scholars have contemplated the linking issue on a more technical level.[178] The question is when a piece of software actually "uses" open source code, i.e. code covered by the GPL, in order for the copyleft provisions to apply to the whole or a combination of the two works. As stated earlier, understanding this is key for any company using open source components in their products or services. Usually this comes down to the issue of *static and dynamic linking*.

Although the question of the type of linking should always be a legal one rather than a technical one, some US courts have categorically defined certain linking types as triggers for ascertaining whether the linking work is a derivative work of the linked work.[179] This might be considered as taking the easy and most convenient way out. Consequently, this is why software companies categorically instruct their developers to properly isolate

[175] Välimäki 2005.
[176] http://www.gnu.org/licenses/gpl-faq.html#MereAggregation, last visited 27.5.2014.
[177] Välimäki 2005.
[178] See e.g. Rosen 2004.
[179] On US Courts' decisions commentary and further legal analysis see e.g. Stoltz 2005.

open source components from their own software components and even not to use static linking methods.

In order to present the matter with more accuracy, concepts of static and dynamic linking should be explained in more detail. Linking occurs when a software program "calls" another program which is not contained in the source code of the calling program.[180] In this sense, *static linking* means that "a calling program is linked to a called program in a single executable module".[181] This usually results in an executable (i.e. file with .EXE extension) or a dynamic link library (i.e. file with .DLL extension) which includes the executable source code for both the called and the calling program. One of the most recognized advantages of the static linking method is that the program is certain that all the needed libraries are present in the executable file, hence making this linking method less dependent on external changes.[182] Furthermore, distribution and installation is more compact as the program may be contained in one executable file as opposed to multiple which, consequently, facilitates the management of the whole.

As opposed to static linking, *dynamic linking* allows several programs to use an executable module and provides for building several subprograms into a DLL-file.[183] Consequently, static linking provides for lighter programs (executables) as linked external programs are able to use the executable without having to be built into it and the behavior of executables of the calling program can easily be changed by relinking them to different external programs without having to make changes to the calling program. Furthermore, if more than one calling program needs access to the linked (calling) program, dynamic linking enables all calling programs to use the same source as opposed to static linking, where duplicates of the programs would be needed. In this sense, DLL-files are, for instance, used to implement subsystems and interfaces to other programs.

From this background it is somewhat understandable that US Courts have considered a statically linked program a derivative work of the called program as, after linking, the object code executable file (i.e. machine readable code) contains code from both programs, making it almost impossible to distinguish them from each other. Stoltz

---

[180] IBM COBOL Programming Guide, available at: http://pic.dhe.ibm.com/infocenter/ratdevz/v8r5/index.jsp?topic=%2Fcom.ibm.etools.cbl.win.doc%2Ftopics%2Figy7pg10.htm, last visited 6.6.2014.
[181] IBM COBOL Programming Guide.
[182] On common advantages, see e.g. http://en.wikipedia.org/wiki/Static_library, last visited 6.6.2014.
[183] IBM COBOL Programming Guide.

compares the above result to adding a chapter to a copyrighted novel and republishing it.[184] Furthermore, Stoltz provides another example where a programmer writes a driver to control a printer and statically links the driver with the Linux kernel, which is licensed under version 2 of the GPL. According to Stolz, static linking between the two programs would almost certainly trigger the copyleft obligations and therefore would need to be licensed under the GPL, as the new program would combine parts of the two programs into one executable file.[185]

As Stoltz's comparison might be apt for some cases, it certainly does not work for cases where the calling program consists of substantially more source code (i.e. the calling program is larger) than the called program. From a copyright law perspective, the called program might not even constitute a work protected by copyright law in the first place as it could merely be, for instance a library, an API or a plugin. This has been laid out by Stoltz as well, although it should be pointed out that this remake refers to copyright infringement rather than the scope of what is considered a derivative work, as he points out that courts have established in some cases that literal copying of e.g. interface information is not infringing, either because "it is *de minimis* or an unprotectable idea".[186]

Rosen claims that the GPL doesn't actually provide any meaningful rules on linking, hinting that the linking method doesn't actually matter.[187] However, and to the contrary, as Rosen's opinions pre-date the adoption of the GPL v.3, specific provision on the linking method was included in Section 1 of the GPL v.3. Clearly an attempt to mitigate legal uncertainty, at least in cases where a program is dynamically linked, libraries not trigger the copyleft obligations.[188] According to the said section, if a work is linked to a GPL library, that work will have to be "specifically designed" to require the linked GPL library to trigger the copyleft obligations. Although some uncertainty still persists, this can be interpreted in a way that software which is designed to work with libraries, the copyleft

---

[184] Stoltz 2005 p. 1462.
[185] Stolz 2005 p. 1450.
[186] Stoltz 2005 p. 1464.
[187] Rosen 2004 p. 116
[188] Excerpt from Section of the GPL v.3: "The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work." Available at http://opensource.org/licenses/GPL-3.0, last visited 18.6.2014.

obligations of the GPL v.3 would not apply if that software is dynamically linked to GPL v.3 libraries. This section of the GPL v.3 might even be interpreted in a way which would mean, as dynamic linking to GPL libraries does not trigger the copyleft obligations when a program is specifically designed to work with libraries, every other kind of linking would be considered to trigger such obligations.

It could be argued that the programs are not interconnected in terms of copyright law but merely connected through a common, if not predominant and often obligatory, technical method of linking. To support this interpretation, leading software gurus, such as Linus Torvalds, have claimed that software which is merely combined with Linux is not subject to the copyleft obligations regardless of the method of linking.[189]

The case of dynamic linking is far more complicated as the case for static linking as a dynamically linked module doesn't intermingle with another program until the user executes it. Therefore, the stand-alone module, prior to connecting or calling to another program, has not been, at least at the offset and automatically, been considered a derivative work as it might merely be derivative for the relatively small part of amount of interface information it has to mechanically copy from the called program.[190] In addition, according to Stoltz, the calling program might also be a derivative work if its only possible purpose is to be combined with a particular program. Stoltz acknowledges, however, that the exact factors for defining when a program is considered a derivative work of another program remain utmost unclear.[191]

As demonstrated by the abundance of legal theories and court-introduced tests regarding the question of derivative works in software, it is perfectly understandable that software companies feel that their position is insecure in regard to using open source software in their business. Furthermore, even if the above might provide some insights on how individual cases should be interpreted, it is far too vague for companies to rely on a general level, i.e. companies will merely refrain from all acts which might trigger the copyleft obligations.

However, from a copyright law perspective it seems almost frivolous and somewhat arbitrary to base the assessment on whether a program is a derivative of another

---

[189] The Linux kernel is licensed under version 2 of the GPL. Example from Rosen 2004 p. 116.
[190] This has been established in the US Court case of Sega versus Accolade, 977 F.2d 1510, 1524 n.7, 9th Circuit, 1992. For commentary and further legal analysis of the case see Stoltz 2005 p. 1451-1452.
[191] Stoltz 2005 p. 1451.

program solely on the technical linking method. Although it is understandable that US courts have come up with various tests in order to facilitate assessment in individual cases, the assessment shouldn't primarily be based on the technical linking method as this kind of interpretation might lead to technical specifications dictating the scope of copyright.

From the point of view of open source community, another consequence of a pure technical approach adopted by the US courts might be that, as static linking may not at least automatically lead to the conclusion that the calling program is a derivative work of the called program, the copyleft provisions of the GPL might not apply in the very cases they are meant for. For instance, companies could keep secret the source code for improvements for open source software, hence making the modifications no longer available to all.[192] It should be noted that linking in the context described above is different from linking copyrightable works in general. For instance, conclusions from e.g. ECJ's recent Svensson case[193] cannot be generalized for linking in software as it only applies to internet links. Furthermore, the legal problems in the Svensson case are substantially different from what has been discussed above. However, some parallels could be established as in the case of SAS Institute the ECJ explicitly stated that although the copying of functionality is not infringing the author's copyright under the Software Copyright Directive, it might be under the Infosoc Directive[194].

3.4.3 Defining Distribution

Besides copyleft and the difficulty of defining a derivative work in the context of the GPL, defining distribution makes for an important step in identifying the underlying risks in open source licenses as distribution triggers the license obligations. According to Niiranen, this is usually where license compliance becomes important.[195]

According to the Infosoc Directive, the author has the exclusive right to authorize or prohibit any form of distribution to the public by sale or by otherwise.[196] This right applies to both the original and copies thereof.

---

[192] Stoltz 2005 p. 1451.
[193] Svensson et al versus Retriever Sverige Ab, Case C-466/12, 13.2.2014.
[194] Directive 2011/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonization of certain aspects of copyright and related rights in the information society.
[195] Niiranen 2013 p. 5.
[196] Article 4 of the Infosoc Directive.

Distribution might at first seem irrelevant to define as it might seem obvious that the licensed software is distributed once it is shipped as is or as part of another software, either as separate works or joint works. This might well be the case for most at least smaller software companies but as a company might operate internationally through subsidiaries and affiliates, the company should consider whether an assignment including open source software to an affiliate abroad might constitute distribution and hence trigger the license obligations.

In addition, the manner in which a company distributes its software might make a significant difference on how and to what extent the copyleft obligations are extended. For instance, as Niiranen points out, if a company distributes software directly to its customers without offering it publicly, it only needs to disclose the source to such individual customers, as opposed to having to disclose them publicly, which would be the case should the company e.g. distribute its software online.[197]

To be clear, in GPL v.3 the license obligations are triggered by "conveying" rather than "distribution".[198] However, in legal literature the term "distribution" is widely used as it is an established legal term and hence covers a more specific content than "conveying". The definition in GPL v.3 in its entirety:

> "*To convey a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying*."

The GPL expressly states that "mere interaction with a user through a computer network, with no transfer of a copy, is not conveying". Consequently, cloud based services which may run on open source software would not trigger the copyleft obligations. According to Meeker, this is the result of deliberation in order to have a consistent definition of distribution compared to mandatory US law which requires actual transfer of a copy.[199] Whatever the origins or the idea behind the definition, it should be borne in mind that, as none of the major open source licenses have provisions of choice of law and jurisdiction, the notion of distribution will be ultimately defined by national courts. The applicable

---

[197] Niiranen 2013 p. 7.
[198] GPL v.3, Article 0: "To convey a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying." Available at: http://opensource.org/licenses/GPL-3.0, last visited 2.6.2014.
[199] Meeker 2012 p. 32.

national law will have to be determined on a case-by-case basis as the according to the rules of private international law of the jurisdiction where the legal action is brought.[200]

Meeker has identified the following problematic scenarios regarding distribution when using open source licenses in connection with:[201]

- Employees
- Independent contractors, i.e.
    - Individuals
    - Consulting companies
    - Outsourcing companies
- Subsidiaries and affiliates
- Mergers and acquisitions
- Productization

As some of the above might seem redundant merely from a practical point of view (e.g. the case of employees), only a few most interesting ones are looked into here. Furthermore, a literal interpretation of distribution is in most cases regarding company's internal arrangements somewhat counterintuitive as the software has not been released to the public.

Although distribution among company's own employees is a simple question, as no distribution occurs since the company is a single legal entity, the question becomes directly more absorbing when third party *consultants and outsourcing* are involved. As outsourcing companies are separate companies and hence the open source software would be distributed.

However, as an alternative method of outsourcing, a company may, for instance, use leased staff to work on its equipment on its premises. In the latter case, at least according to Meeker, the company might be able to claim that the outsourced staff is under its

[200] For more see e.g. Haapanen 2011 p. 250-251.
[201] Meeker 2012 p. 32-34.

control and supervision and no copies of works have been released (or communicated to the public).[202] For this purpose, the GPL v.3 includes an express permission to convey covered works, i.e. distribute software under the GPL, for the sole purpose of having third parties make modifications exclusively for the original licensee, as long as such third party makes such modifications solely for the original licensee's benefit, under its direction and control.[203]

However, for parts of the source code that the original licensee doesn't own, it must give the third party contractor full rights to modify and distribute the code. Other parts of the code, i.e. parts owned by the original licensee and parts generated by the third party contractor on behalf of the licensee, shall not be subject to the obligations otherwise triggered by distribution.

The question of whether providing software to company's *subsidiaries and affiliates* constitutes distribution might depend on e.g. the ownership structure of the company such as whether the affiliate or subsidiary is wholly owned by the parent company as well as on where and how the copy of the open source software is provided. According to Meeker, it might be viable for a company to claim that the copy, although provided to the subsidiary, is merely a private copy of the company group.[204] Naturally, the strength of this argumentation is directly linked with the share of ownership in such subsidiaries.

By *productization*, Meeker refers to the common problem of defining distribution in cloud based business models. According to Meeker, a company may only deploy cloud based software solutions, and, in order to do that successfully, it will have to avoid open source licenses which have explicit provisions on copyleft obligations applying to cloud based distribution.[205] It might well be that, even though the cloud platform is usually deployed solely from the company's servers, some client scenarios might demand, for e.g.

---

[202] Meeker 2012 p. 33.

[203] Relevant excerpt of Section 2 of the GPL v.3: " You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. *You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright.* Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you." Available at http://opensource.org/licenses/GPL-3.0, last visited 18.6.2014.

[204] Meeker 2012 p. 33.

[205] Meeker 2012 p. 34.

regulated security reasons or the like, that the service is operated from the client's premises and on its equipment.

In this case, the company, if diligent on its open source obligations, might not be able to, at least within reasonable time and effort, provide the service demanded by the client as the company has been relying on not having to comply with the copyleft obligations as it in most cases does not apply, and might not be able to provide a similar service as it might have intermixed open source and commercial source code in a way which cannot be reasonably backtracked.[206]

3.5 Legal Consequences of Non-Compliance

Even though the contents of this subchapter is to identify the legal risks, commercial risks are often bundled in the same debate. According to Bain, in a company's risk analysis of using open source software, the views of the free software community as a whole should be taken into consideration, as they seem to have been adopted as a sort of a trade custom or a variant thereof.[207]

As the free software community might be vociferous, although not necessarily always legally correct in all of its assessments of e.g. the implications of license terms, it will pay off to comply with such "trade customs" in order not to cause badwill in the community. Many large software companies, such as Microsoft, who fought the very idea of open source in the late 1990s and early 2000s, have been forced to invest significant amounts of resources in modifying their image in the free software community.

From a legal point of view, as a rule of thumb, the risks software companies seem most troubled with regarding non-compliance with open source licenses are two-fold.

The Infringement Risk:

Firstly, companies acknowledge that the source code distributed under an open source license might infringe third party intellectual property rights. According to Meeker, this problem is representative of the open source development model which allows for everyone to contribute.[208] Consequently, especially in projects where source code due diligence is not conducted, some infringing code might be involved in the source code of

[206] Meeker 2012 p. 34.
[207] Bain 2010 p. 169.
[208] Meeker 2008 p. 151.

the project. This becomes a legal obstacle in particular to software companies which use or embed open source software in their distributable products. Ultimately a company may never know for certain who the correct copyright holder is even though it might be complying with its license obligations. However, this does not exclude the company's liability for copyright infringement but it may limit its liability as to the infringement as it may not have been intentional or grossly negligent. This could have a significant effect on the amount of damages.

The Compliance Risk:

Secondly, and more importantly, software companies face the risk of being non-compliant with the obligations in the respective open source licenses. It is always up to the right holder to decide whether he wants to pursue license non-compliance as a breach of contract or copyright (or other applicable IPR) infringement. This question is, however, irrelevant from a company's point of view as it has no means of controlling the right holders' actions other than by ensuring maximum compliance with the license obligations.

Meeker submits that most companies' biggest fears have to do with being compelled to lay open their own commercial software.[209] Although no court cases exist where an open source software licensee would have been compelled to license its entire work on open source license terms simply because of not having complied with its license obligations, until a competent court in a major jurisdiction rules otherwise, this risk remains, at least on a theoretical level, a true one. At least until now, as a case has been filed in a US court claiming that a work, which has open source components, should in its entirety be licensed under the GPL.[210] However, the argumentation in this case is most likely only used in order to leverage a better settlement from the defendants and it is very unlikely that the court will ever render a judgment on the matter.

Having stated the above, it should be borne in mind that, as presented earlier, most enforcement cases relate to making the court confirm that the licensee in fact complies with e.g. its copyleft obligations by disclosing source code rather than claiming the whole

---

[209] Meeker 2012 p. 277.

[210] Versata Software Inc Versus Ameriprise Financial Inc, US District Court for the Western District of Texas, Case A-14-CA-12-SS, filed March 11, 2014. For more information on the case, see e.g. http://opensource.com/law/14/7/lawsuit-threatens-break-new-ground-gpl-and-software-licensing-issues?sc_cid=70160000000dgMrAAI&elq=5809b8113db847ad97142b5557a3571e&elqCampaignId=31321, last visited 23.9.2014.

work should be licensed under open source license terms. As Meeker puts it, the likely worst case scenario for the licensee is that it will be given a choice either to comply with the open source license obligations or stop using the licensed software.[211]

The above outcome seems the most likely and most logical way to address the problem from the courts' point of view as, contractually speaking, the obligations of a particular open source license have been violated, licensee is in material breach of its contractual obligations. Consequently, the courts can order the licensee to comply with the license obligations or order the licensee to cease its use of the license. In addition, and all other legal aspects aside, courts would likely be more than hesitant to issue an order which would force the licensee to lay open all connected software. Ultimately, what the courts decide depends on the exact contents of the licensor's enforcement claims.

The above reasoning has direct support from the wording of most of the major open source licenses. The GPL, for instance, states that the license is terminated automatically if the licensee breaches its obligations by "propagating and modifying a covered work as expressly provided under this license".[212] Section 4 of the older GPL (v2) has a similar provision according to which "any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License".[213] According to Radcliffe, as the GPL (or any other open source

---

[211] Meeker 2012 p. 277.

[212] Section 8 of the GPL V3: " You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10."

213 Section 4 of the GPL v2: You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

license for that matter) is a conditional license, failure to comply with its obligations automatically terminates the license and, without a license, the licensee becomes a copyright infringer.[214]

Moreover, Rosen claims that even though the meaning of e.g. the GPL is probably to make the open source license govern all subsequent and pre-existing components, it might not necessarily be possible as the author might not have sufficient rights (i.e. copyright) to control those components. Furthermore, Rosen claims that the method of linking should never be a factor when assessing whether a work is based on another work as such linkages merely create collective works.[215] Consequently, all rights to individual software remain under the copyright of the original author.

The above excerpts clearly suggest that, as the license is automatically terminated upon non-compliance, the contractual relationship between the licensor and the licensee has, consequently, been terminated, making it merely impossible for the licensor to claim in court that other license terms, such as the copyleft obligations, should be enforced. However, claims for damages may be possible.

In conclusion, the logical outcome for a software company from the above is that although no legal precedents exist, the actual risk of being forced to license proprietary or commercial software involuntarily, as opposed to merely ceasing use of the open source license at hand, is relatively small.

However, for software companies having built and intermixed open source software with commercial software, the requirement to immediately cease use of open source component(s) might in practice turn out to be impossible. This could mean significant amount of work and resources, as the whole product might have to be rebuilt in a manner which will not trigger the strong copyleft obligations of the source licenses in use or to replace the relevant components with commercial ones. This is an issue which may and should be averted through proper risk planning and guidance, examples of which are presented in the following chapter.

According to Meeker, the resources alone for investigating and settling an open source license claim should be reason enough for a software company to have and maintain a

[214] Radcliffe 2011.
[215] Rosen 2004 p. 119.

robust open source compliance policy as such claims might have IPR related issued as well.[216]

[216] Meeker 2012 p. 290.

4. ADDRESSING AND CONTROLLING RISKS IN AN OPEN SOURCE POLICY

Having previously identified and analyzed the most common legal risks and associated problems with using open source software in their business, this chapter aims at providing general guidelines to software companies on how to control and implement proper guidance on open source use.

This chapter will start by presenting reasoning for a software company, or any other company using open source software for that matter, why it is in their best interest to introduce and maintain a simple and easy to understand open source policy.

According to Tsotsorin, the reasoning for a company to have an open source policy is quite simple, in order to quickly remedy any violations and address any claims in a credible way.[217] Although open source claims might not categorically be as fierce as intellectual property rights claims tend to be (mostly due to the fact that open source software claims are often made by foundations governing their development projects whereas IPR claims are made by entities or IPR trolls who regard their IPR as their core assets and therefore act more aggressively), they have proven to work as most of them are simply a matter of license compliance and not e.g. claims for damages or injunctive relief.

As stated earlier, settling claims before they become formal may have serious implications on the company's goodwill. For instance, a company considering itself an advocate of the open source community and relying heavily on open source software, a formal claim might cause irreparable harm to the company's public image. It might ultimately result in customer mistrust as they might regard the vendor's open source know-how as insufficient.

Furthermore, other reasons for introducing and enforcing an open source policy include, firstly, ensuring that the company knows and is in agreement about how to use open source software.[218] Secondly, the company should always know why it uses open source

[217] Tsotsorin 2013 p. 601.
[218] Peters 2009.

in order to identify the areas where the benefits are the greatest to ensure that the processes will enable company's employees to use open source software effectively.[219]

Ultimately, as every other type of compliance, open source compliance is about having proper directions in order to minimize non-compliance and, consequently, to avoid claims in the first place. Should the licensors never police compliance, companies would have no incentives to comply with open source license obligations in the first place. In this sense, Meeker points out that an essential part of an open source policy is to have the company's employees react quickly to complaints regarding open source software and to take such claims seriously, as they are, more often than not, made through informal channels, such as email to individual employees.[220] This point of view can easily be understood just by observing the minimal number of court cases globally, taking into consideration the ubiquitous nature and sheer number of open source software used worldwide.

As Ballardini has demonstrated in her empirical study about coping mechanisms for legal issues related to the use of open source, most companies having taken part in the study had some sort of compliance and licenses compatibility checks included for the open source software they use.[221] The medium and large companies all had compliance procedures in place in order to assure compliance with license obligations whereas even smaller companies had at least devoted personnel resources in order to assess license compliance.[222]

Since open source advocates, such as the Free Software Foundation and the Open Source Initiative make up most of the potential claimants regarding open source license enforcement, addressing such claims in a timely manner will most likely remedy non-compliance and avoid any further legal action.[223] Furthermore, Tsotsorin points out that most informal claims can be settled on terms that are favorable to all parties involved, as the licensor may well be willing to license the open source component for commercial purposes for a sufficient payment.[224]

---

[219] Peters 2009.
[220] Meeker 2012 p. 290.
[221] Ballardini 2011 p. 22.
[222] Ballardini 2011 p. 22.
[223] Meeker 2012 p. 291.
[224] Tsotsorin 2013 p. 602.

Moreover, a company has numerous alternatives in settling an informal claim, depending on how well it is informed of its use of open source licenses, such as rewriting the code involved, releasing the source code under the proper open source license, ceasing use completely or possibly restricting use of the open source software in question only for internal purposes (i.e. ceasing distribution as it is almost always the trigger for the copyleft obligations).[225]

4.1 Planning the Structure of an Open Source Policy

As most companies use third party software both internally and externally, such as distributing them along with their own software, they should have clear rules, i.e. a policy for monitoring and guiding what aspects related to use of open source are considered acceptable for the company's business purposes and which are not.

Rather than copying an existing policy from another company, a company should always develop their own policy within their organization in order to fit it in with existing internal control mechanisms, such as IPR, innovation and procurement policies.[226] For instance, acquiring open source software differs vastly from acquiring commercial software, at least when the software is acquired directly from the author or an open source project. Kemp stresses the importance of having a few key principles when acquiring open source software. The company should always consider at least the following[227]:

> (i) source reliability: know where the open source software your organization is using is coming from;
> (ii) acquisition: know what open source software your organization is using;
> (iii) tracking: know what the open source software your organization is using does and where it is being used and re-used;

Firstly, the above list summarizes open source compliance concisely as a company should always be aware where its software is coming from in order to assess and possibly address future copyright (and patent) infringement and license compliance claims. Secondly, a company should always be aware and up to date what open source software it is already using and whether such software is used or incorporated in its own products and services. Thirdly, a company should know what the used open source

---

[225] Tsotsorin 2013 p. 602.
[226] Copenhaver 2009 p. 143.
[227] Kemp 2009 p. 64. Full list also includes two additional points:
"(iv) roles and responsibilities: know who is responsible for what; and (v) license compliance: know that your organisation is complying with its FLOSS licence obligations".

software actually does and how it works and where else it is possibly being used. Ultimately, the more information the company has on the software it is using, the better the understanding of the underlying risks should be. Therefore a policy stressing and demanding sufficient due diligence and information gathering on open source software in use in the company should, at least to a substantial extent, alleviate these concerns.

Copenhaver divides compliance programs into two distinct parts, a policy statement and a written internal business process to manage compliance with the policy.[228] A policy statement, which should be available to the public, could be general yet immutable. It could simply be an announcement stating that the company respects the intellectual property rights of others and endeavors not to assert its own IPRs in a hostile manner. In contrast, the internal compliance process should include known best practices and would be subject to change.

According to Copenhaver, involving relevant stakeholders in the creation and adoption of a dedicated policy will result in better buy-in of the personnel and more efficient processes.[229] This is probably true to the extent that employees are more likely to comply with rules in creation and adoption to which they have been involved in, stressing the importance of employees understanding the meaning and function behind such rules. It would only seem logical that rules which are better understood result in higher compliance compared to rules which employees do not comprehend or which the employees consider bureaucratic. Depending on the size and setup of the company, the following stakeholders should be taken into consideration:[230]

- Software developers, as they are the ones having to comply with the policy in practice
- Software architects, as most companies have architectural and technical committees which define their software (and product) architecture
- IT staff, as they probably download and use open source software
- Managers of business teams that use open source software
- Lawyers, in order to assess legal implications of the licensed software
- Chief information officer

Kemp has identified several more stakeholders and given relevant stakeholders objectives and means for achieving those.[231] For instance, the finance team should

[228] Copenhaver 2009 p. 144.
[229] Copenhaver 2009 p. 143.
[230] Example list based on Peters 2009, available at http://www.openlogic.com/wazi/bid/187982/, last visited 25.6.2014.
[231] See table 1 in Kemp 2009 p. 66.

identify relevant open source IPRs as software assets and introduce a book value for them. Moreover, human resources might be involved in identifying relevant aspects for their policies (e.g. what kind of open source contributions are allowed under employees' own name in their spare time).

Copenhaver stresses the importance of having designated individual gatekeepers to approve different aspects of open source use. Furthermore, she poses the question why software companies should bother structuring and breaking down their open source policies into smaller parts.[232] According to her analysis, personal responsibility is key for every part of the process, however bearing in mind that every gatekeeper (i.e. person responsible for making a decision) should have his or her own area of expertise in order to avoid repetition and overlapping analyses on whether use of a certain open source component is feasible for a predefined purpose.[233] Clear responsibilities prevent gatekeepers from revisiting decisions already made by earlier persons. For instance, a lawyer shouldn't revisit or make technical decisions.[234]

Ultimately, governing use of open source software shouldn't be any different from governing use of any other third party software although open source software come with numerous obligations and make compliance somewhat more demanding on a company level. In this sense, the scope of any third party software used, especially if distributed further, need to be checked and assessed in detail before incorporating it into the company's own software. From a policy point of view, creating one-size-fits-all rules for open source software might even be simpler than for commercial third party software as the obligations of open source, according to Tsotsorin, are for most parts identifiable in a uniform manner.[235] This applies at least to certain open source licenses as the license terms are often perpetual and not be changed. Consequently, grouping software under similar licenses and providing guidance for them becomes easier.

What is the option to introducing an open source software policy or using open source in the first place? On the one hand, categorically excluding free and open source software from internal and external use would result in giving the competition an unnecessary edge as almost all companies utilize and benefit from open source software in some way.

---

[232] Copenhaver 2009 p. 145.
[233] Copenhaver 2009 p. 145.
[234] Copenhaver 2009 p. 145.
[235] Tsotsorin 2013 p. 602.

On the other hand, providing all available open source components full legal scrutiny would seem like overkill from a resourcing point of view.

Creating a sensible open source policy should take into consideration the specific needs a company might have for open source software as well as identifying the areas where such use would most likely result in strategic advantage such as cost cutting, reliability and other advantages identified in earlier chapters. However, on a company level, the message on open source use should be as clear as possible. For instance, in order to lower costs the guidance should be "reduce IT costs by using open source instead of commercial software" rather than "use as much open software as possible".[236]

Moreover, the company should identify its strategy for open source use and possibly align and incorporate that strategy with its risk management or IPR strategy in order to verify that the message in consistent. For instance, Kemp provides some useful, albeit mere examples of a company's open source objectives.[237] They include, inter alia, the ability to attract the best talent by building an open source development community at the forefront of open source skills and increasing competitiveness with increased efficiency in development, hence enabling faster time to market and reduced costs.

In addition, Kemp categorizes high-level objectives of an open source policy into four distinct categories[238]:

1. avoiding disputes and managing regulatory risks
2. achieving good management for a financial event – for example, an investment round, IPO, trade sale
3. higher customer satisfaction
4. being a good FLOSS[239] community member

As stated earlier, avoiding disputes and settling them in a timely and satisfactory manner are of utmost importance when setting objectives for an open source policy. This point is also the most important one from a pure legal risk management point of view. Kemp's

---

[236] Peters 2009.
[237] See Table 2 in Kemp 2009 p. 67.
[238] Kemp 2009 p. 63.
[239] FLOSS refers to free, libre and open source software. See e.g. http://en.wikipedia.org/wiki/Alternative_terms_for_free_software, last visited 25.6.2014.

second point merely relates to demonstrating to relevant third parties, such as potential investors or buyers, that the company is diligent in its governance of software and has documented and assessed its use of open source software.

Customer satisfaction, however, is an interesting point of view as with everything else a software company does, also use of open source software should ultimately depend on whether it has a positive effect on the company's customers. For instance, this could mean that the customers get their software products faster, at a lower price or with better interoperability.

The last point, being a good FLOSS community member has more to do with strategic and PR-related aspects as the company may want to position itself as an open source advocate and community benefactor. This might have positive implications on e.g. the talent the company is able to attract. In addition, the company might want to consider open sourcing parts of its software in order to build an online community around it to encourage outside development for e.g. features that the company wouldn't otherwise had known there was a demand for in the first place.[240]

After having identified the objectives for an open source policy, the company should draw up a process which will execute those objectives in practice. In this context, Copenhaver identifies two main types for processes, a vertical process and a horizontal one. A vertical process is initiated through a recommendation made at the operational level which is then passed on to the next persons in the approval sequence, all of whom pass it on to the next person identified in process.[241] The last person in the process is the one ultimately making the decision after having verified that all the previous ones have approved their part. This type of vertical process doesn't leave room for persons in the command line to make alternative proposal or request, hence the decision should always be either to approve or reject.[242] For more practicability, the following illustrates the vertical model[243]:

1. Initiation of a request to use an open source component by a technical employee. Information on the open source component, such as

[240] Peters 2009.
[241] Copenhaver 2009 p. 146.
[242] Copenhaver 2009 p. 146.
[243] Based on examples in Copenhaver 2009 p. 146.

identification of license, open source project, technical feasibility, is gathered.

2. Assessment of usefulness and quality of the open source component and potential benefits of use (such as time or cost savings) by a manager.

3. Assessment of the license terms and their applicability for the proposed use by a legal counsel. Information about business objectives are vital for legal counsel in order to assess license from a correct point of view.

4. Approval by management unit based on earlier information and confirmation that company policies have been followed.

In contrast to the above, Copenhaver's horizontal model provides for decisions to be made on input provided by multiple stakeholders. As a further contrast, other relevant stakeholders are invited to present alternative ways in order to solve the matter at hand.[244] Consequently, the result from a horizontal process may not result in simple approval or rejection, but rather in an entirely new plan where input from multiple stakeholders are taken into consideration.[245] This type of process may include a review board or committee which agrees on the details of the plan in using a certain open source component.

As a summary from the above, some general guidance on how a software company may want to start creating its open source policy can be identified.

1. Identify the company's objectives for using open source software. These can range from time and cost savings to anything between arranging the company's software assets for a merger or an initial public offering.

2. Introduce simple and understandable rules which make it clear to relevant stakeholders what sort of scrutiny is expected from them and which steps are necessary in order to acquire approval for the open source component.

[244] Copenhaver 2009 p. 146.
[245] Copenhaver 2009 p. 146.

3. Set up an approval process which includes personal involvement of relevant stakeholders. Keep in mind the relevant expertise and avoid possible overlaps for stakeholders. Aim for clarity regarding who initiates the process and who makes a decision regarding his/her area of expertise.

4. Keep in mind that processes vary and should always properly reflect the use cases. For instance, an approval process for internal open source may differ from a use case where third party open source code is implemented into a company's products or where such software is otherwise distributed to third parties.

5. Constantly remind and educate employees on the process of introducing open source software into company use. A positive approach in reminding the employees of the company's goals (e.g. more efficient use of third party software software) regarding open source software and how to achieve those in practice might be effective.[246] Also implementing changes and responding to criticism from personnel is key for a successful open source policy.[247]

4.2 Identification and Selection of Open Source Software

Identifying and grouping open source licenses into different categories is key to maintaining an efficient open source policy while providing individual licenses the scrutiny they need. One major classification is whether the open source software at hand is to be used solely for internal purposes or whether it is being incorporated into the company's software products. The external category can further be divided into multiple subcategories depending on what type of products they are meant to be used in.

Identification of different open source licenses and their further categorization seems like a suitable way for a company to identify and categorize its needs and document the use of external open source software accordingly. As internal use is somewhat straightforward since it usually does not trigger the strong copyleft obligations, this part focuses mainly on best practices for external use.

[246] Copenhaver 2009 p. 150.
[247] Copenhaver 2009 p. 150.

Niiranen divides external use of open source software into two subcategories, distribution and offering as a service.[248] Furthermore, Niiranen's open source compliance process consists of three (legal) steps which are i) determining whether the license obligations are triggered in the first place, then ii) identifying the outbound and inbound licenses, iii) determining whether any conflicts between the inbound and outbound licenses exist, and, finally, deciding what actions are needed in order to comply with the applicable licenses.[249] The above use cases are looked into here in more detail.

Internal use shouldn't normally cause any major legal issues as the obligations of the licenses are typically triggered by distribution. Typically a company would, for instance, need to use third party tools for software development. However, as Niiranen points out, it is still important to keep track of the tools used, especially in software development, as they may, intentionally or unintentionally, find their way into products meant for customer distribution.[250]

Distribution of third party open source software defines the legal norm where compliance becomes an actual issue. The exact scope of compliance depends on whether the distributed software has been modified or not. For instance, the GPL v.3 has specific terms for distribution when modification is involved.[251] Should the third party open source software be distributed in unmodified form, compliance should be more straightforward as, typically, the obligations merely include disclosing of proper copyright notices along with applicable and actual license text as well as any source code versions of the software, if applicable. However, some license types include the right to modify the standard license language e.g. relating to any further license permissions, warranty, indemnification requirements for licensors or notification obligations.[252] In this sense, it is important to assess the actual license text distributed along with the software, as it may have modifications, not just to assess the template license text available online.

Compliance becomes much more complicated should the third party open source software be distributed in modified form as then all the copyleft obligations usually apply in their entirety. According to the above method introduced by Niiranen, the company would first have to identify the outbound license. Subsequently, the company would have

---

[248] Niiranen 2013 p. 6.
[249] Niiranen 2013 p. 6.
[250] Niiranen 2013 p. 6.
[251] See e.g. Sections 4 (”Conveying Verbatim Copies”) and 5 (”Conveying Modified Source Versions”) of the GPL v.3. Available at: http://opensource.org/licenses/GPL-3.0, last visited 11.9.2014.
[252] See e.g. Section 7 (“Additional Terms”) of the GPL v.3.

to ensure that the inbound license does not conflict with the outbound license. This means that the company needs to verify that no subcomponents in the distributed software are licensed under a license that would conflict with the outbound license or add any additional obligations to those already imposed by the outbound license.[253] This is often referred to in legal literature as the compatibility problem.[254]

The problem is caused when software or parts of it licensed under different (and strong copyleft) licenses are combined. The classic example of this is the Apache 2.0 and the GPL v.2 as both licenses require that the entire work should be licensed under its license terms. This "battle of forms" has subsequently been recognized. In order to achieve compatibility between the GPL and the Apache license, the Free Software Foundation added provisions to the GPL v.3 (namely patent and optional indemnification) which are required by the Apache 2.0 license.[255] This was understandable, as the goal of the free and open source movement has been the proliferation and worldwide reuse of code and compatibility problems due to legal reasons were not furthering this goal. According to Ballardini, compatibility problems between major open source licenses might, ironically, even have distorted rather than encouraged wide reuse of code.[256] However, this probably applies only to a small part of the potential users of open source software.

Lastly, the company would need to determine what actions are needed in order to comply with the applicable outbound license. This is where the legal problems explained in earlier chapters come into play in practice as this is the point where the legal scrutiny should be conducted at the very latest. Non-compliance might result in IPR infringement or breach of license claim. Should no guideline on inclusion on open source components exist, the damage might be impossible to undo as it might be too late to change the way the open source component has been implemented in the overall solution. This, in turn, might have a serious impact on whether the whole software work should be licensed under an open source license or whether merely the changes in the open source

[253] Niiranen 2013 p. 7.

[254] See e.g. Meeker 2007.

[255] Black Duck Webinar 20.1.2014: "In addition to achieving compatibility with Apache, the new compatibility structure described above allows developers to combine GPLv3 code with code covered by licenses that consist of any of the permitted additional provisions. In the past, developers had to rely on the FSF's statements affirming compatibility with a particular license before related code could be combined with GPL code. With GPLv3, the language itself makes clear that GPLv3 is compatible with a range of licenses, including, for example, the MIT license. That is an improvement over GPLv2, where compatibility was based on industry practice and statements made separately from the licenses themselves".

[256] Ballardini 2012 p. 22.

software should be licensed under the applicable open source license and to be disclosed in source code form.

As a minimum requirement, a company should put into place simple yet effective rules of what is allowed and what is not. For instance, rules regarding use of certain open source licenses might come in handy for a company where different software tools are used. For this purpose, a company might introduce e.g. the following categories of licenses:

- Preapproved licenses
- Approval required
- Prohibited licenses

For instance, a company may have to use different software for research and development purposes than for products or services deployed to customers. Hence, *preapproved licenses* may vary depending on the context and purpose of use, although some licenses, such as the BSD 2.0 or MIT licenses are relatively safe as they merely obligate their licensees to retain relevant copyright notices and original license text.

Licenses that require prior approval might vary greatly between companies as their business logics are different. For instance, some companies may require that the Apache 2.0 license is subject to prior approval from a technology council or IPR board, whereas other companies may well have the same license on their preapproved list of licenses. Others may have it on their preapproved list, provided however that certain base conditions are met, such as not licensing anything under this license for which it has or will apply patent protection for as the license does not allow for patent grant back obligations.

The means of how approval has to be acquired also varies greatly between companies, as in some companies only manager-level approval for certain licenses may be required, whereas in other, typically larger companies, technology boards ultimately decide whether the proposed software should be used or not. Of course, the decision whether software under a certain open source license should be used in the company might have to do with whether it is incorporated into the company's software, whether it is shipped to customers "as is" or whether the software is used only for providing a service to customers via a backend system.

Prohibited licenses should include licenses which may not, under any circumstances, be used or at least not incorporated into company software that is distributed to customers. For some companies this might be GPL v.3 because of its patent non-assertion obligations or the AGPL, in particular for companies offering software as a service, as the AGPL's copyleft obligations would be triggered even when used over a network, i.e. when service is delivered from the cloud.

Moreover, factors affecting the contents of an open source policy vary vastly from company to company. As discussed earlier, as distribution is usually the main trigger for copyleft obligations, a company may need to assess its policy in order to identify problems which may arise when a company has a multiple layer of worldwide teams, subsidiaries and third party contractors all contributing to the company's software development.[257]

Furthermore, a company may want to assess its strategic position in using open source against its competition. For instance, using strong copyleft licenses for e.g. plugins and other types of add-on software which add value to the company's core products, might be a good strategic choice. Consequently, the strong copyleft obligations would prevent the competitors from reaping its benefits without contributing changes under the same license.[258] In this way, the company will be able to benefit from improvements made by the user community, including its competitors. Meeker stresses the business objectives behind each release as it will dictate the choice of license.[259] However, it should be noted that this is only applicable to outbound licensing.

4.3 Managing and Complying with Copyleft Obligations

Even though identifying and categorizing licenses is important in order to create an efficient yet controllable open source policy, it is even more important to understand which rights the licenses provide and what obligations are involved. As open source licenses often curtail the uses for which a company has been used to, it is important to comply with those obligations. As a policy only provides the guidelines, it is up to the relevant personnel to actually comply with and follow the policy. As has been noted

[257] Kemp 2009 p. 64.
[258] Meeker 2008 p. 136.
[259] Meeker 2008 p. 136.

earlier, for this reason it should be eminent that the better the personnel involved with the policy understand the reasoning behind it, the more likely they are to comply with it.

Compliance should be both administrative and technical. *Administrative compliance* includes approval mechanisms in place in order to police and checklist the requirements of the open source policy. These can be put into practice in many ways, e.g. through having managerial approval (with one or more steps) prior to taking the open source software into use. The manager would then have a checklist in order to verify that all the necessary steps have been followed and that the previous managers have checked the items they are responsible for. A process like this could include that the person responsible for presenting the case to the manager would have provided an executive summary for the benefits of using the particular software and a possible alternative, acquired approval from the legal department and comments from a relevant software architect.

For compliance reasons, a company should always try to fit the suggested open source component into its open source policy framework. By this is meant that while the policy should be as simple and lean as possible, taking into consideration that most open source licenses are uniform, there should be enough operating room to introduce licenses which do not necessarily fit into the policy. However, introducing an excessive multitude of non-uniform licenses will pose compliance problems as they won't fit into the normal processes created specifically for the purpose of keeping license compliance at a high level. It would be in a company's best interest to keep the number of non-uniform license at a minimum as they would require bespoke control mechanisms which will increase both cost of compliance and likelihood of non-compliance.

Administrative compliance should also comprise of constant review of the open source policy as e.g. lawyers will have to identify the most risky parts of the licenses from a compliance point of view and the managerial and technical departments will have to consider means to introduce and maintain a system which will ensure an acceptable level of compliance. For most parts, this can be achieved through reporting and monitoring rather than through introducing technical compliance systems which may be a too large an investment for many companies compared to its potential benefits. This is at least the case for many small and medium sized businesses.

Technical compliance might not be a viable option for smaller companies as the costs for running and maintaining a dedicated technical system might quite easily surpass the

benefits of having such technical monitoring. For instance, open source code can be tracked in the company's own source code repository with different software solutions. If open source code is included in the company's own products, an easy yet effective solution would be to instruct software developers to clearly mark any third party code directly in the source code, i.e. the source code itself would clearly indicate where third party source code starts and ends. Even though knowing the exact components the company has incorporated in its products is vital, the obligation of listing components and their related licenses might come from the company's customers as they are following their own open source policy and will therefore have to know this in detail. As listing all included open source components might be a contractual obligation towards the company's customers, not being able to list and detect all open source components along with their licenses might be considered a contractual breach.

Larger companies may want to track all third party source code formally through a dedicated tracking database or through procedures that will dynamically provide all information on third party source code. Furthermore, there is an abundance of different audit and checking software and services for open source license compliance, both free and commercial.[260] For instance, a company may want to use software tools which enable it to select appropriate open source components in an early phase, i.e. during design or pre-development, and scan the existing source code and application base to identify troublesome licenses and their dependencies. Furthermore, a single component may consist of numerous subcomponents which might all be licensed under different licenses. Naturally the company will need to comply with every single one.

As the previous subchapter addressed issues concerning the choice of open source licenses, the next logical step in an open source policy is to establish the ground rules for use of such accepted licenses. A good rule of thumb might be, at least for software which is meant for distribution, to always prefer commercial versions rather than open source versions. This goes for dual-licensed open source components which are licensed both under an open source license (and most likely strong copyleft licenses in order to steer customers with commercial needs for the license towards the commercial version) and a commercial license. However, this would mean that some of the benefits of using open source software in the first place might be lost.

---

[260] For different audit and scanning solutions available for tracking open source software, see e.g. http://www.openlogic.com/solutions/open-source-scanning/, last visited 30.9.2014.

The reasoning for this sort of rule would be to minimize the amount of strong copyleft licenses in the company's distributed software or services, unless its associated cost savings are substantial. Cost varies largely between available open source components and in some cases the commercial version might be a very affordable alternative taking into consideration the risks and compliance cost associated with distributing a component licensed under a strong copyleft license. This is why the company needs to be able to calculate the TCO involved with different types of licensing options.

Secondly, a company can greatly affect what kind of legal risk it is willing to take with open source software. For instance, a company should introduce clear boundaries within which stong copyleft licenses can be used. As stated earlier, for most companies it is probably best that the most troublesome licenses, such as the GPL v.3 and the AGPL are always subject to individual technical and legal scrutiny. However, for other coplyeft licenses, such as the GPL v.2, a company may well grant authorization without e.g. legal scrutiny, provided that some clear conditions are fulfilled. Use should always be subject to satisfaction of some predefined criteria, such as that the component is to be licensed to third parties and that no commercial software option is available for a reasonable fee. Provided that the preconditions are met, the following rules might be an efficient way to control the legal risk:

1. Isolate the open source component as an independent component that operates and functions independently from other company software.

2. Retain all copyright and license notices in the source code.

3. Make source code available and possible modifications thereto freely available to all.

The first point addresses the problem of derivative works which has been presented and analyzed earlier. In order to prevent the copyleft obligations from applying, a company might want to take necessary technical measures in order to isolate the distributed open source component from the company's commercial software. For other software than libraries, this would in practice mean that the policy explicitly forbids its developers to use either static or dynamic linking as a linking method to the open source component.

This is to prevent the company's software from being applied as derivative works of the open source component, i.e. a single work in the eyes of the open source license. The

company should design its software in a manner which will reduce risk when its software is used together with open source software. For instance, all core functions of the software should be the company's own IPR, whereas open source additions to the company's software might be restricted to value adding modular features. Modularity would be key as components would then be less complicated to replace as such without having to change the core architecture of the company's software. This is of course not feasible in all cases and the company needs to revise its policy accordingly. This way of building software is widely acceptable and even instructed by the open source community.[261] Be it as it may, the open source component should always be incorporated in a manner which makes it technically feasible to replace it if needed. For specific types of software, such as libraries, some bespoke and less stringent directions may be needed. As an example, dynamic linking may be allowed, depending on the license.[262]

The second and third point are in most cases quite simple obligations to comply with from a legal point of view, but may turn out to be cumbersome from a technical point of view in case their implementation has not been taken into consideration from the very beginning. It may very well turn out that including copyright notices or license text will result in unnecessary additional work when deadlines approach. This is where the open source policy comes into play as it will, if successfully incorporated into the company's processes, do its part in streamlining and cutting costs when it is being followed correctly.

[261] See e.g. http://www.gnu.org/licenses/gpl-faq.html#GPLInProprietarySystem, last visited 30.9.2014.
[262] For instance, the LGPL v.2 allows for dynamic linking without triggering the copyleft obligations of the license.

5. Conclusions

In this final chapter I will to discuss and further conclude the aspects discussed earlier in this thesis. Firstly, I will put open source licenses in their general copyright and IPR context as incentives for innovation. Secondly, I will sum up the findings of the most relevant legal concepts in open source licenses. Furthermore, I will briefly discuss the enforceability of open source licenses in general. Thirdly, I will go through the legal implications of license non-compliance. Finally I will tie together the benefits of having an open source policy as a legal risk management tool.

As already pointed out in this thesis, copyright was chosen as a means of protection for software mainly for practical reasons. On the one hand, copyright was as existing means of protection and with minor amendments it was fitted as such for software. On the other hand, copyright, as it manifests in the idea-expression dichotomy, only provides for limited IPR protection for software as the most valuable parts (i.e. parts for which a company has invested the most monetary and personnel resources) might not be protected at all as they are often just an underlying technical method in order to reach a certain form of expression. Therefore it could be argued that copyright has failed its purpose for software as right holders need to invest in additional means of protection in order to yield benefits (i.e. get the planned return on investment) from their limited monopoly. Consequently, right holders seek for additional IPR protection e.g. though patenting and trade secrets.

In this sense, the copyright system does not deliver the results it was initially created for as source code is often concealed and regarded as a trade secret by the right holders, hence not being able to provide information and inspiration to the public in return for the right holders' limited monopoly rights. Accordingly, it might not be a far-fetched argument that the alleged shortcomings of the copyright system have influenced the adoption of alternative legal instruments, such as open source licenses, in order to apply copyright protection for software the way it was initially planned. This is in line with the motivations of the free and open source movement as its agenda is to promulgate software freedom in general by pointing out inherent flaws in the IPR system. Its most vocal proponents even advocate for the total abolition of IPRs in software.

The key concepts of open source licenses from a legal point of view are, as presented in this thesis, copyleft, distribution and the derivative works question. Although copyleft

is not an established legal term, it carries well established obligations for the licensees of open source software. Understanding its boundaries and its relation to derivative works is paramount in order to even attempt a conclusion of the legal obligations related to open source licenses.

As the scope of a derivative work is always defined on a national level, it remains yet to be seen how (or if) courts in European jurisdictions will address the question. It is however not very likely that major litigation would arise in the near future as right holders, as well as licensees are somewhat content with the current state of affairs. Accordingly, the status quo regarding open source litigation is that the major players in the industry as well the open source movement (as major right holders in many significant open source projects) seem to value the current state of things more than finding out and trying the boundaries of open source obligations in court. However, some conclusions can be made although legal precedents are not of abundance. In the US, courts have created their own tests in order to assess whether a work is derivative. These tests are, as presented in this thesis, very technical in nature, and seem to work contrary to what was initially intended for copyright protection. In my opinion, the nature of e.g. linking (whether static or dynamic) can hardly be the correct legal tool in order assess whether a work is derivative or not and hence covered by the open source license. It would be surprising if a European court would take a stance in favour of either linking method over the general concepts of copyright law, i.e. similarity of the works in the form they are expressed.

Moreover, it has been demonstrated in this thesis that open source licenses as such are enforceable agreements between the licensor and the licensee. The licensee accepts the obligations of the license by using the software. From a Finnish perspective, as the licenses are conditional agreements, it very much depends whether the general rules on contract formation have been fulfilled. As these rules vary from country by country, the exact legal implications of a license is always done on a national level and depending on the circumstances of a particular case. Whereas a license obligation might be enforceable in a certain situation, it might necessarily not be enforceable in another. At the onset, software companies would have a hard time convincing a court in whatever jurisdiction that a valid agreement has not been formed and, consequently, the company would be released from honoring its license obligations. However, it should be reminded that most obligations in open source licenses are only triggered upon distribution.

Companies and other licensees alike dread the thought of open source license non-compliance. However, the findings of this thesis does not offer direct support for the most serious fears as licensees will, in practice, have a possibility to remedy their non-compliance in multiple ways. A licensee which is in breach of its license obligations might e.g. cease using the open source component in question, stop distributing the component or merely render its use of the license compliant. However, the most likely outcome of license non-compliance is termination of license and possible damages for copyright or other applicable IPR infringement, not strict enforcement of copyleft obligations (i.e. forcing licensee's derivative work to be licensed under an open source license).

From a legal policy point of view, although copyright legislation is general in nature, it might not be an utterly groundless thought to introduce specific rules which would take into account e.g. different licensing instruments, such as open source. Introduction of specific rules would increase legal certainty without modifying fundamental aspects of copyright law even if they were to merely state and reinforce the current state of affairs. However, introduction of too specific rules might not be a good idea as software and its related business models keep on developing at a fast pace. Furthermore, it should at least be considered that the current laws on copyright protection for software (in the absence of other applicable IPR protection) do not protect the investment of the right holders in a satisfactory manner.

As a final conclusion of this thesis the benefits of having an open source policy could not be stressed enough. Companies relying on open source software should be well aware of the obligations embodied in open source licenses. A company should always perform its due diligence as to what open source components it is using and whether the license carry significant obligations. An open source policy should be considered as a risk management mechanism which purpose is to, on the hand, lay out the and facilitate the circumstances and conditions under which a company can yield the benefits of open source software in the most effective way, and, on the other hand, prevent the company from breaching its license obligations and hence become subject to a legal claim.

REFERENCES

Aarnio, Aulis
Laintulkinnan teoria. WSOY 1989 (Aarnio 1989).

Asay, Matt
Never a Better Time for Open Source. International Free and Open Source Software Law Review, Volume 2, Issue 2, 2010 (Asay 2010).

Bain, Malcolm
Software Interactions and the GNU General Public License, International Free and Open Source Software Law Review, Volume 2, Issue 2, 2010 (Bain 2010).

Bainbridge, David
Legal Protection of Computer Software. Tottel Publishing 2008 (Bainbridge 2008).

Ballardini, Rosa Maria
Intellectual Property Protection for Computer Programs – Developments, Challenges, and Pressures for Change. Helsinki 2012 (Ballardini 2012).

Brown, Vance Franklin
Incompability of Copyright and Computer Software: An Economic Evaluation and a Proposal for a Marketplace Solution. North Carolina Law Review 66, 1987-1988 (Brown 1987).

Carver, Brian W.
Share and Share Alike: Understanding and Enforcing Open Source and Free Software Licenses.
Berkeley Technology Law Journal Volume 20, 2005 p. 443-481 (Carver 2005).

Chesbrough, Henry W
The Era of Open Innovation. *Sloan Management Review*, 44, 3, Spring 2003 (Chesbrough 2003).

Christie, Andrew
Designing Appropriate Protection for Computer Programs. European Intellectual Property Review 1994, 16(11), p. 486-493 (Christie 1994).

Deek, Fadi P. – McHugh, James A.M.
Open Source – Technology and Policy. Cambridge University Press 2008 (Deek-McHugh 2008).

Dolmans, Maurits
A Tale of Two Tragedies – A plea for open standards. International Free and Open Source Software Law Review, Volume 2, Issue 2, 2010 (Dolmans 2010).

Evans, David S. – Hagiu, Andrei – Schmalensee, Richard
Invisible Engines. How Software Platforms Drive Innovation and Transform Industries. MIT Press, Cambridge Massachusetts. London, England 2006 (Evans et al 2006).

Ferguson, Doug

Syntax Error: Why Version 3 of the GNU General Public License Needs Debugging. North Carolina Journal of Law and Technology, Volume 7, Issue 2, Spring 2006 (Ferguson 2006).

Haapanen, Anna
GPL Embedded in Devices - 10 Most Important Questions for Device Manufacturers' Consideration. European Intellectual Property Review, 33(4), 2011, p. 245-251 (Haapanen 2011).

Heller, Michael A. – Eisenberg, Rebecca S.
Can Patents Deter Innovation? The Anticommons in Biomedical Research. Science 280, 698, 1998 (Heller – Eisenberg 1998). Available at: http://www.sciencemag.org/content/280/5364/698.full.pdf, last visited 28.10.2014.

Hietanen, Herkko
The Pursuit of Efficient Copyright Licensing. How Some Rights Reserved Attempts to Solve the Problems of All Rights Reserved. Doctoral thesis, University of Technology, Lappeenranta, 2008 (Hietanen 2008).

Kemp, Richard
Towards Free/Libre Open Source Software ("FLOSS") Governance in the Organisation, International Free and Open Source Software Law Review, Volume 1, Issue 2, 2009 (Kemp 2009).

Mann, Ronald J.
Commercializing Open Source Software: Do Property Rights Still Matter?
Harvard Journal of Law & Technology, Volume 20 Number 1 Fall 2006 p.1-48 (Mann 2006).

Meeker, Heather: The Open Source Alternative. Understanding Risks and Leveraging Opportunities. John Wiley & Sons, 2008 (Meeker 2008).

Meeker, Heather: Open Source and the Age of Enforcement. Hastings Science & Technology Law Journal, Vol 4:2 Summer 2012 (Meeker 2012).

Mustonen, Mikko
Copyleft – The Economics of Linux and Open Source Software.
Information Economics and Policy 15 (1) 2003 p. 99-121 (Mustonen 2003).

Niiranen, Ossi: Open Source Software in Your Business – Controlling the Risks and Reaping the Rewards (Niiranen 2013). Available at: http://documents.jdsupra.com/2c30065d-1276-4f8a-88d8-2fd6c6084d2d.pdf, last visited 10.10.2014.

Landes, William M. & Posner, Richard A.
An Economic Analysis of Copyright Law. Journal of Legal Studies 1989, p. 325–363 (Landes-Posner 1989).

Perens, Bruce
Innovation Goes Public.
Computer and Telecommunications Law Review 14 (2) 2008, p. 36-40 (Perens 2008).

Peters, Stormy

Best Practices for Creating an Open Source Policy, 2009 (Peter 2009)
Available at http://www.openlogic.com/wazi/bid/187982/, last visited 24.9.2014

Radcliffe, Mark
Legal Challenges in Android Development, Law Technology News, May 31, 2011 (Radcliffe 2011).
Available at:
http://www.lawtechnologynews.com/id=1202495473435?slreturn=20140517081740, last visited 28.10.2014

Ravicher, Dan
Software Derivative Work: A Circuit Dependent Determination, 2002 (Ravicher 2002)
Available at:
http://www.pbwt.com/Attorney/files/ravicher_1.pdf, last visited 9.10.2014

Raymond, Eric S.
The Cathedral and the Bazaar.
2nd edition. O'Reilly 2001 (Raymond 2001).

Rehm, Robert J. Jr., Navigating the Open Source Minefield: What's a Business To Do? 10 Wake Forest Intell. Prop. L.J. 289, 314-17, 2010 (Rehm 2010).

Rosen, Lawrence
Open Source Licensing: Software Freedom and Intellectual Property Law, Prentice Hall 2004 (Rosen 2004).

Rowe, Kirk D, Why Pay for What's Free? Minimizing the Patent Threat to Free and Open Source Software, 7 J. Marshall Rev. Intell. Prop. L 595, 607, 2008 (Rowe 2008).

Samuelson, Pamela
Should Economics Play a Role in Copyright Law and Policy. University of Ottawa Law & Technology Journal 1, 2003-2004 (Samuelson 2003-2004).

Stallman, Richard M.
The GNU Operating System and the Free Software Movement.
In: Di Bona, Chris – Ockman, Sam – Stone, Mark
Open Sources: Voices from the Revolution 1999 p. 53-56 (Stallman 1999).

Stoltz, Mitchell L.
The Penguin Paradox: How the Scope of Derivative Works in Copyright Affects the Effectiveness of the GNU GPL, Boston University Law Review Vol. 85:1439, 2005 (Stoltz 2005).

Tsotsorin, Maxim V: Open Source Software Compliance: The Devil Is not so Black as He Is Painted. Santa Clara Computer and High Technology Law Journal, April 2013 (Tsotsorin 2013).

Tritton, Guy – Davis, Richard – Edenborough, Michael – Graham, James – Malynicz, Simon – Roughton, Ashley
Intellectual Property in Europe. Sweet & Maxwell London 2008 (Tritton et al 2008).

Van den Brande, Ywein – Coughlan, Shane – Jaeger, Till (Edited by)

The International Free and Open Source Software Law Book. Open Source Press 2014 (Van den Brande et al 2014).

Von Willebrand, Martin
Case law report: A look at EDU 4 v. AFPA, also known as the “Paris GPL case”. International Free and Open Source Software Law Review, Volume 1, Issue 2, 2009 (Von Willebrand 2009).

Välimäki, Mikko
Oikeudet tietokoneohjelmistoihin. Helsinki 2009 (Välimäki 2009).

Välimäki, Mikko
GNU General Public License and the Distribution of Derivative Works', The Journal of Information, Law and Technology, (1), 2005 (Välimäki 2005).

Välimäki, Mikko
Dual Licensing in Open Source Software Industry. Systèmes d'Information et Management, Vol. 8 No. 1, pp. 63-75, 2003 (Välimäki 2003).

Webbink, Mark
Packaging Open Source, International Free and Open Source Software Law Review, Volume 1, Issue 2, 2009 (Webbink 2009).

Westkamp, Guido
The Limits of Open Source: Lawful User Rights, Exhaustion and Co-Existence with Copyright Law.
Intellectual Property Quarterly (1) 2008, 14-57 (Westkamp 2008).

Williams, Sam
Free as in Freedom 2.0 – Richard Stallman and the Free Software Revolution. Free Software Foundation, Boston 2010 (Williams 2010).