Best Practices of Adopting Open Source Software in Closed Source Software Products

Diplomarbeit im Fach Informatik

vorgelegt von

Martin Helmreich

geb. 09.07.1984 in Nürnberg

angefertigt am

Institut für Informatik
Professur für Open Source Software
Friedrich-Alexander-Universität Erlangen–Nürnberg

Betreuer: Prof. Dr. Dirk Riehle

Abgabe der Arbeit: 31.03.2011

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 31.03.2011

Martin Helmreich

Diplomarbeit/ Master thesis

Topic:

Best Practices of Adopting open source software in Products

Summary:

A software vendor can save money, speed up development, and increase product quality by using offtheshelf open source components in its products. However, embedding open source comes with a risk: If not handled properly, the vendor may have to open source its source code, may be sued for patent infringement, or may end up depending on an open source community that acts against its interests. This thesis analyses these threat scenarios and collects best practices for coping with them.

Problem and Work Description:

Many open source components are of high quality yet (appear to be) free to get. Linux, Apache, MySQL, etc. (i.e. the whole LAMP stack) are good examples. It may seem obvious then that developers of software packages like business applications or manufacturers of hardware devices that use a lot of software employ open source software to speed up development, improve product quality, and save money. A couple of factors speak against this, however. For example, "viral" open source code may require the vendor to open source its own source code. Or, the developers of an open source component may decide on a direction that does not fit the vendor's interests. This thesis analyses the situation, determines the top three threat scenarios from the vendor's perspective, and collects best practices for coping with these threats. The best practices are analyzed, streamlined, and integrated into a suggested process for adopting open source software in a software vendor's products.

Thesis Advisor:

Prof. Dr. Dirk Riehle dirk.riehle@informatik.uni-erlangen.de

Abstract

Open source software offers great chances in software development through cost savings and shorter time to market. Open source software is used in companies – in some cases conscious, in others unconscious – and puts its implications on the business.
This thesis deals with these implications. Threats a company might face are outlined and best practices to encounter them are named.
This was performed by a combination of research in publications and interviews with employees at companies that deal with open source components.
Result of this research is on the one hand a set of threat scenarios and on the other hand, a set of best practices a company should follow to avoid problems arising from the usage of open source software.
Finally, it can be stated that open source software can be handled without putting the business at risk. The number of practices to follow and actions to perform is manageable.

Contents

1 Introduction

Open source software is omnipresent.[26]

Most PC users will have used (perhaps not knowingly) open source software like Firefox, Thunderbird, Open Office, GIMP, VLC, Linux, Wordpress or Apache. In December 2010 Apache holds a market share of nearly 60%.[19]

As well as in end user applications open source has become important in application development. Forrester analyst Jeffrey S. Hammond states that "Adopting open source software isn't a question of if but a matter of how [15]."

Sojer et al. found in a study that 15% to 21% of the interviewed software developers have used open source software disregarding license obligations. They found that such inappropriate use of open source software is influenced by the developers' attitude and perceived subjective norm, which is in turn influenced by the ethical work climate. The severity of consequences (for the individual developer as well as for the company) and the severity of time pressure also influence the attitude to such inappropriate usage.[27]

Related Work

This thesis is based on several existing papers that either discuss practices or offer ideas for practices.

The Forrester survey performed by Jeffrey S. Hammond suggests some best practices to perform for open source adoption. It is based on the statements of fifteen companies and prior Forrester publications. There are mentioned several good ideas, but these could be affirmed through neither the other examined works nor the performed interviews.[15]

The DLA Piper presentation, due to the company being a law firm, has a legal view on the topic. It addresses many aspects, but outlines none in detail. [24]

Similar to the DLA Piper presentation the BigFix presentation has its focus on legal aspects, especially license issues. The covered subjects are comprehensive, but not handled in-depth.[2]

A few selected, but good and detailed issues are described in the IBM presentation. It addresses mainly process and policy issues, but does not mention any threat scenarios.[29]

The examined OpenLogic webinars share the presence of prominent amount of advertisement for the company's products, which had to be neglected. Also, there are only evanescent few threat scenarios mentioned.
The webinar "Six Steps to open source License Compliance" is – as the name suggests – focused on licenses.[35]
In "Ten Key Elements of open source Governance in the Enterprise", there are non-technical high-level business aspects brought up that provide a broad overview.[20]
The excerpt "5 Roadblocks to open source Adoption" from the webinar "Top 10 Ways to Stretch Your Budget by Using More open source software in 2010" is kept in a general context and mentions different acceptance thresholds to open source software adoption.[34]

The OpenLogic Certification Process characterizes the actions that are performed before an open source component is included in their open source repository.[23]

Guidance how to create and install an open source policy is provided by the "Navica open source Policy Process". Aside policy there are some practices that are related to infrastructure aspects.[14]

As the caption of the Black Duck Whitepaper "The Business Case for Automating open source Code Management" indicates it deals with infrastructure aspects how the process can be automated. There are also depicted some threat scenarios.[4]

The Whitepaper "Seven Best Practices for Managing Software Intellectual Property in an open source World" presents seven selected practices in detail.[5]

Contributions and Context

The contributions of this work are:

- Threat scenarios companies might face when adopting open source software
- Best Practices – based on papers as well as interviews – that facilitate companies to adopt open source software without exposure to unnecessary risks

The context of the work is:

- The research was performed in a qualitative manner

- The interviewees worked at internationally operating software companies
- The interviews were performed in German language

Structure of the work

This thesis will research how open source software can be adapted in commercial products. It will consider aspects relevant for software development, not for usage of open source applications.

- Chapter 2 gives an overview what open source means and outlines general risks and benefits of open source software.
- An overview over the intellectual property rights can be found in chapter 3.
- The implications by these rights on open source software discusses Chapter 4, section 4.4 is about open source licenses.
- In chapter 5, the Methodology used for the following analysis is described.
- For each analyzed document, a summary is written in chapter 6.
- Every performed interview is recapitulated in chapter 7.
- The threat scenarios that were extracted from literature and interviews are presented in chapter 8.
- Finally, chapter 9 shows the best practices that were developed from literature and interviews.

2 Open Source

2.1 Definition of Open Source

Open source software (OSS) is Software that is licensed under an open source license. A license is an open source license, when it complies to the requirements that are stated by the open source Definition (OSD) which is maintained by the open source Initiative (OSI).[21] The OSD requires a license to implement ten conditions to be named "open source License". Each condition is illustrated with an example from the GPL.[10]

2.1.1 Free Redistribution

The license must not restrict redistribution by prohibiting contribution of the software in an aggregate along with software from other sources or by claiming royalty or other fees.
GPL: "[...] 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. [...]" (§5) "[...] You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty [...]" (§10)

2.1.2 Source Code

The source code must be distributed along with the software. It must be allowed to distribute the source code as well as the compiled program. If some sort of product is distributed without source code (consider anything with firmware), there must be an easy way (preferably as free download in the internet) to obtain the source code. In this case, there may be charged at most the reproduction costs. The source code must be in a programmer-friendly version. It is not allowed to deform the source code (e.g. putting all source code in just one line). The output of a preprocessor or the like does not meet this requirement.
GPL: "You may convey verbatim copies of the Program's source code as you receive it [...]" (§4). "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 [...]" (§5). "You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License [...]" (§6).

2.1.3 Derived Works

It must be allowed to modify the program and to create a derived work. It must be allowed to distribute the new work under the same license as the original work. Everybody who received the software under an open source license is allowed to distribute any modified or derived work of this under the same license.
GPL: "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 [...] You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. [...]" (§5).

2.1.4 Integrity of The Author's Source Code

To protect the author's source code, the license may prohibit distribution of modified source code, but only if it allows distribution of "patch files" that allow modification at build time. Distribution of software built from modified source must be allowed as well. The license may require a new version number or name for the derived or modified work.
GPL: The GPL allows to add a clause to achieve "[...] Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version [...]" (§7)

2.1.5 No Discrimination Against Persons or Groups

The license must not exclude anyone or any group from the granted rights.
GPL: The GPL does not discriminate any person.

2.1.6 No Discrimination Against Fields of Endeavor

The license must allow usage of the program in any field of endeavor. So it is not allowed, for example, to prohibit use in military surroundings.
GPL: The GPL does not discriminate any field of endeavor.

2.1.7 Distribution of License

When a program is redistributed, everybody who receives the program must get the rights that were attached to the program without execution of an additional license.

GPL: "[...] You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. [...]" (§5)

2.1.8 License Must Not Be Specific to a Product

The rights granted by the license must be granted autonomous. It is not allowed to grant the rights only if the program is distributed as a part of a particular distribution. All users are conceded the same rights, even if the program was extracted from the original distribution.
GPL: With the GPL it is not possible to grant rights only if the software is distributed within a distribution.

2.1.9 License Must Not Restrict Other Software

The license must not set any requirements about other software that is distributed together with the open source software. Therefore, it is not allowed to prohibit distribution of other (even proprietary) software on the same medium.
GPL: "Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate." (§5)

2.1.10 License Must Be Technology-Neutral

No clause of the license may be based on a specific technology or interface.
GPL: The GPL is technology-neutral.

2.2 Risks

The use of open source software holds some risks. These risks are not as serious as one may suppose because the most reservations are based on non-knowledge. They are just different from risks created by commercial proprietary software. The mentioned risks address inclusion of open source software in products, not usage of open source.

2.2.1 Hidden costs

When the process of evaluation of the OSS is skipped or disregarded, you may end up with an OSS that does not meet your requirements. The contrary, but just as disadvantageous scenario evolves when the evaluation is repeated by different people or teams due to lack of a central depository of approved OSS. During your product's life cycle (or even later) you may be confronted with legal and compliance issues (see chapter 3, "Intellectual Property"). Lawsuits are often cost intensive and finally you may be sentenced to indemnification.[33]

2.2.2 Professional support

Usually there is no explicit manufacturer of OSS, so there will be no professional support—as it is usual for proprietary software—by the manufacturer. There are companies that offer support, but there is nobody who will support "his own" product.[22]

2.2.3 Missing longevity

Missing longevity may create new challenges. You have to look for alternatives if an OS component you adopted in your product is ceased. But this also may happen with a commercial software (may be less likely and less abruptly).[22]

2.2.4 Volatility

Often open source projects are very volatile, it is not unusual having more than one release per month. There is additional effort to keep track of such fast developing projects and to integrate the new releases in your product.[22]

2.2.5 Training efforts

In the end-user environment there may be higher training efforts because users may be more familiar with some particular proprietary software. This aspect has less impact in development because developers have to understand proprietary software components as well, but the documentation of proprietary components may be better.[25]

2.2.6 Community politics

There may arise problems if the community of an OSS you use changes its (political) direction. In consequence, the ongoing development will not meet your requirements anymore.

2.2.7 Abandon IP rights

When you contribute to an open source project, you will inevitably abandon (parts of) your intellectual property rights.

2.3 Benefits

Usage of open source software provides many benefits that are worth dealing with the risks listed in chapter 2.2.

2.3.1 Bug fixing/ reliability

As there is no bug-free software, the issue of fixing bugs is important. In traditional commercial software development, the developers of the manufacturer are the only people who can access the source code. So any bug can be fixed only by them. Even effective and efficient search for bugs is restricted to the developers. Users can just report a phenomenon of unexpected or unusual behavior of a program and wait for the reaction of the manufacturer.

As the source code of an OSS is available for everyone, a lot more people can investigate about the nature of an unexpected behavior and can (given enough skills) provide a patch that can be distributed to the other users. Furthermore, the availability of the source code allows an in-depth investigation for possible bugs that have not been visible through a phenomenon observable by a user. For these reasons bugs in OSS tend to be detected and fixed more quickly than those in proprietary software.[12, 37]

The "State of Software Security Report" from Veracode shows that open source software is as secure as commercial software, but more secure than internally developed software. The time until a security vulnerability in open source software is fixed (36 days) is significant shorter than the time that elapses until a commercial product gets fixed (82 days). Another advantage of open source is the "relative absence" of backdoors.[32]

2.3.2 Security

Some people claim that closed source software is more secure than open source software because attackers cannot use the source code to search for possible vulnerabilities. On the first sight, this seems to be an important difference, but it disregards the existence of decompilers. While the source code produced by decompilers is pretty useless to enhance the program due to missing variable names, comments and so on, it can be used to discover patterns that indicate possible security holes. So unknown leaks are to be considered time bombs for open source as well as for proprietary software. A crucial difference is that any finder of a vulnerability in a close source program has no possibility to fix it. In contrast, users of OSS are able - given capability and resources - to write a fix and to commit it to the open source project. Having numerous people looking on the code the change rises that somebody discovers a security hole and it gets fixed. Of course, not all programmers will examine the code in-depth, but there are two types of programmer who will: first those, who want to use the software personally

and second those, who work for a company that relies on that software and requires a security audit by its information-security policy. As a side effect these many eyes looking on the code force developers to write their code more clear and compliant to standards. One problem affects both, open and closed source software: you cannot be sure whether a review has taken place. Just the possibility to review code does not imply a review has been done. OSS can easily be more secure than proprietary software, but it is no automatism. Security requires an active community that provides and distributes fixes for security vulnerabilities. In one case, there should be paid special attention: when closed source gets open source. Due to the inevitably existent security holes in the software and the non-existence of a grown community there will possibly be a lot of leaks be discovered, but not fixed right away.[36, 37]

2.3.3 Customization

In commercial and educational environments, there often is a request for an individual version of specific software. When an institution has chosen proprietary software, it has generally very limited possibilities to adapt the software to its individual needs. To have any bigger changes implemented they will have to persuade the manufacturer to implement these requirements and in a row pay for it the price set by the manufacturer. When chosen an OSS, the institution is able to implement all desired changes on its own or hire a developer. In this way the changes can be done possibly more individual and at lower costs. A facet of customization is translation into less common languages. Proprietary software is often available only in widespread languages and there is no possibility to get an adapted version. OSS can easily (despite the workforce needed) be translated in any desired language.[37]

2.3.4 Avoiding lock-in

An organization is "locked in" to a specific software product, if the costs for switching to an alternative product are so high that a migration to another software product is strictly avoided. Such a situation arises when an organization uses a non-free product with non-open formats in a highly integrated manner. In this situation, there are two scenarios to consider: First, software is just a tool that was acquired to do a job. As long as there are no significant changes in process, there is no need to change the software. A software product can be in use over several years without wearing out. This stands in contrast to the objectives of a software manufacturer or vendor. They want to sell software or at least upgrades for their software. When an organization uses a specific product intensely enough, it's easy to sell (even unneeded) updates by changing to a new (of course better) file format or stopping provision of critical bug fixes. Second, it is usual that processes change. When an organization is very fixated on a specific software product it can become difficult to adjust the process in the way they want to. The solution that improves the process may be constrained by the software used. OSS

avoids such lock-in rather autonomously. Generally, OSS uses open standards and can therefore easily be replaced by other software that uses that standard. Even if an open source component uses an own format that is non-standard, due to the available source code the format is quasi open.[12, 37]
Furthermore the performed interviews showed that companies contribute to open source projects to evolve their own original proprietary formats towards a standard.

2.3.5 Costs

OSS can generally be obtained at zero cost. There is also no need for an accounting of copies (=licenses) in use. Of course besides other costs, e.g. for training and support, incur, but they also do when using proprietary software. It is difficult to get information about the TCO of OSS in comparison to proprietary software.[37]
The performed interviews showed that companies appreciate the savings made by not developing or buying commodity software components.

3 Intellectual Property

Intellectual property law protects harvest of creative work. It is divided in three rights:

Copyright protects the form of expression

Patents protect methods

Trademarks protect names and signs.

3.1 Copyright

Copyright protects "original works of authorship" including texts, music, videos, pictures and other intellectual works. Protection is independent of publication of the work, even unpublished schemes are protected. The author is the only person who has the right to copy, distribute, modify or publicly show the work. Copyright protects the form of expression, not the subject matter. Given an existing implementation of an algorithm there would be an infringement, if someone copies the code, but there is no infringement if someone just adopts the algorithm and elaborates a new implementation that has the same effects as the original, but is textual different. The only criterion for protection is "intellectual creation by the author". Other aspects as aesthetics or quality do prevent a work being protected by copyright law. Exempted from protection are only banal elements that every programmer would use in the same or a similar way. The "Berne Convention for the Protection of Literary and Artistic Works" (short: Berne Convention) is an international contract signed by 164 nations that harmonizes international copyright law. All member-countries of the Berne convention guarantee to protect the author's copyright inside and outside the country of origin and grant the rights in most cases[1] for at least fifty years after the author's death. The protection takes effect automatically, no formal request is required. After expiration of this period, a work becomes "public domain" *(German: gemeinfrei)*. In some countries (e.g. the USA), authors can make a work public domain and forfeit all rights, in other countries (e.g. Germany) this is not possible, only (unlimited) right of use can be conceded. Transference of copyright is also important in employment context. During an employment all commercial relevant rights (the extent is dependent from national law) are automatically transferred from

[1] In case of anonymous or cinematic works the protection may expire fifty years after publication, in case of photographs 25 years after creation.

the employee who created the work to the employer who "bought" the service "create work" by paying wage.[7, 30, 38]

3.2 Patents

Patents protect inventions. A patent grants "the right to exclude others from making, using, offering for sale, or selling the invention". Nobody is allowed - independent of knowing about the patent - to do anything based on the protected invention without the patent holder's permission. There are three conditions to be fulfilled if an invention is intended to be patented: First, the invention has to be new, i.e. it may not be "prior art". "Prior art" covers everything that has been published in any way in the past. Research whether the idea is prior art or not is the most extensive part of a patent approval because "any" publication of the same idea disqualifies the idea for a patent. Second, the idea has to be an "inventive step" respectively "non-obvious" (different terms used in different countries). An expert in this field must not be able to get the same idea in an obvious way.
Third, the idea has to be usable in a commercial way. This is almost always given.
When a patent is granted, all patent documents are published and there is possibility for third parties to raise caveat or revocation action.
A special discussion is held about software patents. In Germany, for example, software per se is not patentable. However, there rose the problem that inventions that contain software were excluded from patent protection. That means a machine that does its job just by mechanics is patentable, but a machine that does the same job (partially) by software is not patentable. In consequence, also software (at least firmware) can become patented. The distinction between mere software programs and inventions containing software is difficult. Another (now and then funny) problem is so-called "trivial patents". A patent is called trivial if someone managed to get a patent on a very simple, common used method. Often such trivial patents do not survive a lawsuit.[7, 30]

3.3 Trademarks

Trademark law is the least problematic intellectual property right because it is very easy to investigate whether someone has protected a specific trademark. Trademarks (more precisely: trade- and servicemarks) protect signs (logos, phrases or combinations) that are attached to products. There are two ways to acquire protection by trademark law: first, you can apply for enrollment in the register at the trademark office. If you are the first who uses the desired sign you get enrolled for a rather small fee and your sign is protected as a trademark. This is the most common way. Second, a sign can implicit gain protection my trademark law, when the sign is used widely and has become "secondary meaning".[7, 30]

4 OSS and IP

As open source software is like all software creative work it is affected by intellectual property law. This chapter shows what impact the three rights have on open source software.

4.1 Copyright

Some special considerations about copyright are associated with OSS. In traditional software development, there is one author of the software product: A company or a private. It is obvious who holds the copyright. In OS development there are many - possibly hundreds or thousands - authors of one piece of software. Due to the partial sophisticated regulations of copyright law, someone who wants to use or even modify the software or its source code has to ask all authors for permission to do so. It is obvious that this is hardly feasible. To encounter this problem two facilities developed over the time: OS licenses and foundations. Licenses provide standardized conventions; foundations maintain OS projects (and licenses) and can act in court as "the author" to sue copyright infringers.

4.2 Patents

In general, OSS benefits from the exclusion of private and experimental use from patent protection. Everybody is allowed to use patented inventions in a non-commercial manner without infringing patent protection. So it is commonly possible to use patented source code legally in OSS, but once the software is distributed in a commercial way, the private/ experimental use exclusion does not take effect any more. The same applies of course if OS is developed by a commercial organization.

4.3 Trademarks

There is no OS-specific impact from trademarks. All assigned trademarks can be researched in public lists. OSS can infringe trademark law as well as all other software, too. This is easy to avoid because you just have to search in sorted lists for the desired name of your product.

4.4 OS licenses

Open source licenses deal with these intellectual property rights (usually mainly Copyright and patents) and set up regulations about that.

4.4.1 Categorization

OS licenses are categorized based on two criteria:

1. When a modified version of the original verison is distributed, the source code of the modification has to be made available to everyone who received the binary.[6]
2. A derived work of the software may be combined only with software licensed under the same license.[6]

Permissive

None of the two criteria is met. Software licensed under a permissive license can be easily adopted; there is no pressure to publish any source code. Such software can be integrated in commercial, proprietary products without revealing any secrets. Licensing under this license means to donate (rather) unlimited usage right to any- and everybody. Examples: BSD-license, Apache-license

Copyleft

The first criterion is met: Publication of the modified parts Software licensed under a copyleft license can be adopted, but due to the publication of modified parts there may be revealed some secrets, at least concerning the interface. Licensing under this license means to donate extensive usage right to any- and everybody, but also to prevent anybody to bag the profits in private realized on the author's code. Examples: GNU Lesser General Public License (LGPL), Mozilla Public License (MPL), Eclipse Public License (EPL)

Viral copyleft

Both criteria are met: modifications have to be published and the software may be combined only with identically licensed software. Software licensed under a viral copyleft license is problematic in business environment. When such software is adopted, every code that is combined with it has to be published under the same license. If done so, every secret in any piece of code related to the adopted software will be revealed. Licensing under this license ensures that the source code and everything based on it will be free forever. Example: GNU General Public License (GPL)

4.4.2 Viral copyleft licenses

4.4.2.1 GNU General Public License

The GNU General Public license (GPL)[10] is the most widely used OS license.[3] It is rather complex and contains some sophisticated rules. Here is an aggregation, sorted by paragraphs to facilitate look-up.

0. Definitions

To "modify" means any form of copying or adapting, despite making an 1:1 copy.
To "propagate" means any action that is relevant in copyright meanings, including copying, distributing, making available to the public.
To "convey" means all kinds of propagation that makes receiving or making copies possible for a third party.
In version 2 the word "distribute" was used, but now replaced by "propagate" and "convey" because "distribute" has in some countries already a predefined meaning in copyright law that was different from the meaning in GPL v2. The new words prevent that misunderstanding.

1. Source Code

In this paragraph the extent of the source code covered by the license is defined (called "Corresponding Source". It means any source code that is "needed to generate, install, and (for an executable work) run the object code and to modify the work", excluding the "System Libraries". Unfortunately, there is no exact (technical) description, what the System Libraries include. The GPL enumerates "kernel, window system, and so on" (sic!) as well as compiler and object code generator.

At this point there is no mutual consent, which programming techniques qualify for a System Library. This is especially interesting if someone plans to integrate GPL-licensed source code in a proprietary code. The Free Software Foundation states that in this case only communication "at arm's-length" is allowed, otherwise the two programs would be seen as parts of one combined work which would be covered by GPL.

There are other opinions that linking is also allowed. But even concerning linking there are two points of view: Some say only static linking is allowed, others are convinced that even dynamic linking would create a combined work.

2. Basic Permissions

It is allowed to "make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force." If you convey covered

works to other for "the sole purpose of having them make modifications exclusively for you (under your direction and control), or provide you with facilities for running those works", there are no conditions for the parts for which you control copyright. For the other parts (for which you do not control copyright), you have to comply with the conditions of this License.

3. Protecting Users' Legal Rights From Anti-Circumvention Law

This paragraph is about "effective technological measure" that are used to enforce copyright, e.g. copy protection of a CD or Digital Rights Management (DRM).
It is not possible to use software licensed under GPL as an effective technological measure. Even if it would be effective, it is declared not to be, so it is no litigation to circumvent this measure. In this way, the GPL does not restrict what people do with GPL-licensed software, but it prohibits restriction.
The author waives any legal power to forbid circumvention of an effective technological measure.
This paragraph is new in version 3 since the WIPO Copyright Treaty and its corresponding article 11 were adopted in 1996.

4. Conveying Verbatim Copies

It is allowed to convey 1:1 copies, if no copyright or warranty notice is removed, the copy carries a prominent copyright notice and a copy of the license is enclosed.
There may be charged a price for each copy or offered support or warranty.

5. Conveying Modified Source Versions

When conveying a "work based on the Program, or the modifications to produce it from the Program" in source code form four conditions have to be met:

a. "The work must carry prominent notices stating that you modified it, and giving a relevant date."

b. "The work must carry prominent notices stating that it is released under this License and any conditions added under section 7."

c. The license applies always to the whole work, unconcerned the packaging. The work cannot be licensed under another license. An already existing other license is not touched.

d. Every interactive user interface must carry legal notices, but you need not to retrofit missing notices in existing interfaces.

If the work is put together with other independent works in an aggregate (that may be a storage medium, ZIP-file, etc.) the license does not apply to the other works of the aggregate.

6. Conveying Non-Source Forms

When you convey the work in object code form (under the conditions in No. 4 and 5) you have to convey the source code in one of the following ways, too:

a. Conveying the object code in a physical medium
   i. Enclose the source code on a physical medium (e.g. CD, DVD)
   ii. Enclose a written offer to
      1. give anyone who possesses the object code a copy of the source code on a physical medium for a price not higher than your one expenses for the physical conveying.
      2. access the source code on a network server at no charge.
   iii. In case you received the object code with a written offer and you act noncommercial, you may convey individual copies with a copy of the offer.

b. Conveying the object code "by offering access from a designated place (gratis or for a charge)", you have to offer access to the source code in the same way at no further charge. Object and source code may be on different network servers, but you have to "maintain clear directions next to the object code" how to find the source code.

c. Conveying the object code using peer-to-peer-transmission, you have to inform the peers where the source is available to the general public at no charge.

It follows a sophisticated description of the conditions that apply when object code is conveyed in a User Product (embedded software). This is new in version 3 and was initiated by "tivoization[1]".

7. Additional Terms

With version 3 of the GPL you may add additional terms to the license. There are two types of additional terms:

a. Additional permissions may be added to the entire program or parts of it. When you convey the work, you may remove any or all additional permissions or add your own.

[1]Tivoization is named after a digital video recorder sold by TiVo, Inc. They used software licensed under GPL v2 and made the source code available on their website. Thus, they complied with the conditions of GPL v2. In the device they installed a mechanism that checked the digital signature of the installed software. In this way it was nearly impossible to get a modified version run on the device.

b. Non-permissive additional terms are limited to the following:

   i. "Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

   ii. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

   iii. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

   iv. Limiting the use for publicity purposes of names of licensors or authors of the material; or

   v. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

   vi. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors."

Any other non-permissive additional term are called "further restrictions" (see also section 10). When you receive a program licensed under this license, you may remove any further restriction. If you added additional terms to the license, you must place a notice in every affected source file.

8. Termination

"You may not propagate or modify a covered work except as expressly provided under this License." If you anyway do so, all rights granted by this license to you are automatically terminated. Your license is reinstated, if you end all violations of the license. This reinstatement is new in version 3.
Rights other received from you under this license are not affected. If your rights are terminated and not reinstated, you cannot receive any new licenses for the same material.

9. Acceptance Not Required for Having Copies

You can have copies and run the program without accepting the license, but accepting the license is the only way to get the rights for propagating and modifying.

10. Automatic Licensing of Downstream Recipients

With each conveying every recipient automatically receives a license from the original licensors.

"You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License." "[Y]ou may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed".

11. Patents

This paragraph (new in version 3) contains ingenious defenses against patent lawsuits: Everybody who contributes to the work grants all licensees a patent license (including all rights given by this license) concerning all patents that are affected by the version he contributes. If the contributor gives that license only to a few licensors, it is automatically extended to all.
It is not allowed to use knowingly patented code in a contribution if the contributor has not the right to give a patent license to the licensees.

12. No Surrender of Others' Freedom

If you are subject to any obligations that dissent the conditions of this license, the license stays in force. The only possibility to conform to both, the license and your obligations, will probably be to relinquish all rights given by the license.

13. Use with the GNU Affero General Public License

Compatibility clause for version 3 of the GNU Affero General Public License

14. Revised Versions of this License

The different possibilities to handle future versions of the license:

a. If in the work is specified a certain version number followed by the phrase "or any later version" you may use the mentioned version or any later.

b. If no version number is specified in the work, you may choose from all ever published versions.

c. If it is stated that a proxy can decide which version has to be applied, that proxy's public statement sets the version that is to be used.

15. Disclaimer of Warranty

16. Limitation of Liability

17. Interpretation of Sections 15 and 16

Sections 15 and 16 exclude any warranty and liability. In some cases (as in the European Union), this is restricted by local law.

4.4.3 Copyleft licenses

4.4.3.1 GNU Lesser General Public License

The GNU Lesser General Public license (LGPL) [11] is a modification of the GPL. It allows a library being linked by a program licensed under a GPL-incompatible license, even a proprietary one. A prominent example is the GNU C library (glibc).

0. Additional Definitions

"Library" means a work covered by this license.
"Application" means any work that uses the interface of the library.
"Combined Work" means a work where the Application is linked with the Library.

1. Exception to Section 3 of the GNU GPL

Paragraph 1 overrides paragraph 3 of the GPL, thus it's possible to use LGPL licensed code in "effective technological measure" that are used to enforce copyright.

2. Conveying Modified Versions

If you modify the Library in a way that it uses data provided by the Application, you have two possibilities to convey the modified version. Either you ensure that the Library does its job as expected, even if the Application does not provide the data or you license it under the GPL. Licensing the Library under the GPL would mean that the Application has to be licensed under GPL, too.

3. Object Code Incorporating Material from Library Header Files

If your application incorporates material from a header file that is part of the library and you convey the application under a license of your choice, you have to place a prominent notice on each copy that the Library is used and covered by this License. Furthermore, you have to enclose a copy of this license and the GPL.

4. Combined Works

You may convey a combined work under any license if you comply with all of the following conditions:

a. Modification or reverse engineering of the Library part may not be restricted.

b. Give a prominent notice on each copy about the Library and its license.

c. Enclose a copy of the GPL and the LGPL.

d. If during execution a copyright notice is shown, it must contain a notice about the Library and directions to copies of the GPL and LGPL.

e. Enable re-linking with a modified version of the Library by

   i. Conveying the source code that is based on the Library

   ii. Using a shared library mechanism that "uses at run time a copy of the Library already present on the user's computer system, and [...] will operate properly with a modified version of the Library that is interface-compatible with the Linked Version."

f. Provide Installation Information according to section 6 of the GPL.

5. Combined Libraries

If you combine party of the Library licensed under this License with parts of another library licensed not under this License in a single library and you convey this library under a license of your choice, you have to enclose the extracted parts that are based on the Library. Furthermore, you have to give a prominent notice with the combined library where to find the extracted parts.

6. Revised Versions of the GNU Lesser General Public License

The proceeding concerning new versions of the license is analog to that describes in the GPL.

4.4.3.2 Mozilla Public License

The Mozilla Public License (MPL) [18] is relesaed by the Mozilla Foundation.

1. Definitions

"Contributor Version" means "the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor."
"Patent Claims" means "any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor."

2. Source Code License

2.1. The Initial Developer Grant

a) Grant of an all-embracing copyright license concerning the Original Code.

b) Grant of an all-embracing patent license concerning patents that are affected by the Original Code.

c) The licenses in a) and b) become effective with the distribution of the Original Code under this License.

d) The Initial developer grants no patent license for code that is deleted from the Original Code that is separate from the Original Code or for infringements caused by modification or combination of the Original Code.

2.2. Contributor Grant

analog to 2.1

3. Distribution Obligations

3.1. Application of License All modifications are regulated by this License. When you distribute the Source Code version, you have to do it under this license of a future version of it and you must enclose it to every copy you distribute. You may not restrict the rights granted by this license.

3.2. Availability of Source Code Any Modification has to be made available in Source Code under this License. You may enclose the Source Code on the same media as the Executable or make it available online. If you choose the online version, you have to ensure that the Code is available for at least twelve months after it came available or six months after a new version of the Modification became available.

3.3. Description of Modifications All changes have to be documented in a file stating also the date of modification. There has to be included a prominent statement mentioning the Initial Developer as well in the Source Code as in a notice in the Executable version.

3.4. Intellectual Property Matters

(a) Third Party Claims "If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2", Contributor has to put a description of the claim and whom to contact in a file named "LEGAL". If that knowledge is obtained after distribution, the file has to be adapted and all recipients of the code have to be informed about the new knowledge.

(b) Contributor APIs "If Contributor's Modifications include an application programming interface and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file."

(c) Representations "Contributor represents that, except as disclosed pursuant to Section 3.4 (a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License."

3.5. Required Notices You have to put the notice attached in Exhibit A of this license in each source code file, or if this is not possible in the corresponding directory.
Warranty, support, indemnity or liability obligations may be offered for a fee, but those may affect neither any other Contributor nor the Initial Developer.

3.6. Distribution of Executable Versions If all requirement in sections 3.1 - 3.5 are met, the Executable version of the Covered Code may be distributed. This can be done under another license if the other license does conform to this license and no rights in the Source Code granted bay this license are restricted. If the distribution is done under another license it must be clear that this does not affect any other Contributor or the Initial Developer.

3.7. Larger Works The Covered Code may be combined with other code under another license to a Larger Work if the terms of this License are met for the Covered Code.

4. Inability to Comply due to Statute or Regulation

If any statute, juridical order or regulation causes you not to comply with any term of this license, you have to comply to the "maximum extent possible" and describe the limitations in the LEGAL file (see section 3.4).

5. Application of this License

"This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code."

6. Versions of the License

6.1. New Versions The Mozilla Foundation may publish new versions of the License; each will be given a unique version number.

6.2. Effect of New Versions When a new version is available, you may choose whether you use the Covered Code under the version under which the Covered Code was published or under the new version. Only The Mozilla Foundation may change the terms of this License.

6.3. Derivative Works This paragraph describes what to pay heed to when creating or using a modified version of this License.

7. DISCLAIMER OF WARRANTY

8. TERMINATION

8.1. In case of failing to comply with the Licenses all rights granted by this License are automatically terminated. If not cured, 30 days after becoming aware of the infringement. All properly grated sublicenses to the Covered Code will survive.

8.2. This paragraph protects in a sophisticated way from patent lawsuits. In a nutshell: If someone initiates any patent litigation (even if not correlated with patent grants given by this license), the granted rights are terminated or a "mutually agreed" royalty has to be paid.

8.3. If a patent infringement claim is solved before initiating a litigation the value of the licenses granted in section 2.1 and 2.2 shall be acknowledged when determining the amount or value of any payment or license.

8.4. Validly granted end user licenses will survive termination.

4.4.3.3 Eclipse Public License

The Eclipse Public License (EPL) [9] evolved from IBM's Common Public License. (CPL)[16] It is often categorized in the same category aside the GPL, but if you take a close look you will see that it is much more similar to the LGPL, because it allows to incorporate an EPL-licensed Program in a proprietary Software.[8]

1. Definitions

"Contribution" means the initial code, changes and additions to the Program.
"Contributor" means "any person or entity that distributes the Program".
"Program" means the distributed Contributions.
"Recipient" means any person (including Contributors) who receives the Program.

2. Grant of Rights

a. Each Contributor grants Recipient an all-embracing copyright license.

b. Each Contributor grants Recipient an all-embracing patent license concerning all patents affected at the time of the Contribution by the combination of the Contribution with the Program.

c. Each Contributor just grants copyright and patent licenses, but does not provide any assurance about any other patent infringements. The Recipient is solely responsible for complying with any other intellectual property rights.

d. Each Contributor indicates that it has sufficient copyright rights to grant the copyright license specified in this agreement.

3. Requirements

a. A Contributor may distribute the Program in object code, if
   i. It conforms to this license
   ii. Its license
      1. "effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied [...]"
      2. "effectively excludes on behalf of all Contributors all liability for damages [...]"
      3. "states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party [...]"
      4. "states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner [...]".

b. “When the Program is made available in source code form:”

   i. “it must be made available under this Agreement”

   ii. “a copy of this Agreement must be included with each copy of the Program”

4. Commercial Distribution

This paragraph prohibits a non-commercial Contributor being made liable due to assertions a commercial Contributor made.

5. No Warranty

6. Disclaimer of Liability

7. General

If the Recipient institutes a patent lawsuit, alleging that the Program infringes the Recipient's patents the grated rights under 2b are terminated.
If the Recipient does not comply with the terms of this license and does not cure the violation all rights granted by this license are terminated. The Recipient's obligations and granted licenses are not affected.
Only the Agreement Steward is allowed to modify this license. Each new version is assigned a unique version number. A Contributor may decide to distribute under the new version.

4.4.4 Permissive licenses

4.4.4.1 BSD-license

The BSD-license [31] is very short and easy to understand:

```
Copyright (c) <YEAR>, <OWNER>
All rights reserved.

Redistribution and use in source and binary forms, with or
   without modification, are permitted provided that the
   following conditions are met:

    * Redistributions of source code must retain the above
      copyright notice, this list of conditions and the
      following disclaimer.
```

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

Concisely, it could read like this: You can do what you want with the software, but the license text must always be present and do not use my name for advertisement.
Historical note: the license originally contained another condition, which was deleted in 1999:

```
All advertising materials mentioning features or use of this
   software must display the following acknowledgement: This
   product includes software developed by the University of
   California, Berkeley and its contributors.
```

4.4.4.2 Apache-license

The Apache-license [1] is released by the Apache Foundation and grants all copyrights and in addition also patent licenses for patents hold by a contributor. This patent license will terminate automatically when the licensee initiates a lawsuit against any entity claiming a patent infringement by the licensed software. For distribution, the following conditions are set:

- Pass the license text on all recipients of the (modified) software
- Mark all modified files with prominent notices
- Do not remove any legal notice from the source code
- Redistribute the "NOTICE" text file, if included

The modifications or derived works may be licensed under another license, if compatible with the license.

5 Methodology

For this thesis, the following methodology was used:

Literature research Since there is a lot of literature addressing usage of open source applications, but use of open source code in development implies very different obligations, the relevant publications had to be identified.

Categorization: Literature The information in the identified literature was put into categories that were built up during the process. This was based on qualitative content analysis and grounded theory.[13, 17]

Clustering The constructed system of categories was concentrated and clustered, equal and very similar categories put together.

Categorization: Interviews The developed category system was used to categorize the information contained in the interviews, new categories were added.

Clustering Once more, the categories were concentrated and clustered. Generalizations and abstractions were made to reduce the number of categories.

Revisiting Material Literature as well as interviews were revisited with the actual category system. Any inconsistencies created by generalization were removed along with duplicates. This guarantees that if one speaker mentions a topic five times, it is counted only once.

Best Practices Best Practices and Threat Scenarios were evolved from the categories.

6 Related Works

6.1 Forrester Survey

Forrester analysts Jeffrey S. Hammond, John R. Rymer and Justinas Sileikis performed a survey driven by the need to answer the question "How will your organization manage its inevitable adoption of OSS?" They talked to "development managers, architects, analysts, CTOs, and developers". As a result they found three Best Practices:[15]

Create a concise open source software policy

- Specify your goals and aspirations for OSS usage.
  You should make transparent what your goals are and which areas are affected (e.g. development, application software, ...). Integrate people who have already adopted OSS beforehand to increase acceptance of the new policy.

- Involve general counsel early and often.
  Architects and developers in general can tell you only if the OSS component can be used reasonably. There are many other considerations before the component is used for that you had better consult a lawyer. It may be a hard way until the lawyer understands the situation and aids your matter.

- Ensure that developers are able to absorb the information in your policy document.
  KISS: Keep it short and simple. Nobody will read a thick book. Pay attention that the policy gives clear advice on relevant topics.

- Classify software by license type and deployment impact.
  For each license (or license category), you can state whether the license (or category) is allowed, allowed under some restrictions or not allowed. You should also distinguish between different ways of usage. Presumably, there will be no problem if you use a GPL-licensed tool during your testing process, but there might be severe issues if you link a GPL-licensed library.

- Gain an understanding of the potential dangers of distribution and viral licenses.
  Unmindful use of code licensed under a viral license may have evil consequences. Let the legal department approve every license before use.

- Do not treat policy-making as a once-and-done task.
  Open source Licenses are evolving as case law is. Therefore, the OSS policy has to be reviewed regularly, at least once per year.

- Do not prematurely declare victory.
  It is not enough to set up a policy. It is necessary to make it known and followed.

- Do not let lawyers lead the policy-setting process.
  Legal department is just supporting the decision makers in the development department. You have to be careful that discordance between legal and IT does not end up in a "don't ask, don't tell"-strategy by the developers.

Re-engineer the software acquisition process

- Define a consistent rubric to evaluate all types of software.
  The Capability Maturity Model Integration for Acquisition (CMMI-ACQ) provides good suggestions how a mature acquisition process should be organized. It also provides with the "iron triangle" (costs - schedule - capability) an entry point for OSS.

- Make sure evaluations include the entire cost profile of commercial and OSS alternatives.
  Determine the total costs of ownership (TCO) for every piece of software, including costs for license, support, training and upgrade. Differentiate between mandatory and optional costs as well as between fixed and variable costs.

- Reorder acquisition tasks to front-load evaluation and selection.
  You should ensure that definition of requirements, verification and validation of acquired software are performed at the same time and in the inception of the process.

- Demand a realistic assessment of the need for "ilities."[1]
  Separate "good enough" from "nice to have" requirements. Of course, we always want the best, but in most cases, a much simpler (and cheaper) alternative will suit as well.

- Start OSS adoption at the lower levels of the application platform stack.
  As the most widely spread OSS is infrastructure software like operating systems, databases and webservers this is a good point to start OSS adoption.

- Encourage the use of OSS alternatives to drive better deals with conventional suppliers.

[1] scalability, accessibility, reliability, maintainability, extensibility, portability, ...

As soon as you are capable of adopting OSS in a reasonable manner, you can use this in negotiations with your commercial suppliers. If you can point out that an OSS alternative will suit as well, your supplier might make a better offer than before.

- Do not rely on purchase authority as a process control for software acquisition.
  Because OSS needs no management sign-off of an expense report, there might be no notice when an OSS component is used. You have to shift documentation, who is using what software in which environment to your application life-cycle management process.

- Do not overemphasize direct costs simply because they are easier to calculate.
  As said above, there are more costs than license costs. Even if you self-support the OSS there also will be costs. An exhaustive analysis is necessary.

Make targeted adjustments to people, processes, and tools

- Require project leaders to identify OSS dependencies.
  Ensure project managers fill out a "certificate of originality (COO)", stating which software is used in the project code. Do not allow any project without a COO.

- Trust, but verify with code-scanning utilities.
  Use automatic code scanning utilities like Black Duck Protex or Palamida Application Security that identify dependencies and violations.

- Use architects to regulate OSS exploitation and maintenance.
  Architects need special knowledge about OSS and its impacts. For example, on the one hand it may take more time and effort to install and configure OSS libraries compared to commercial counterparts. On the other hand, you may profit from using commodity hardware because there are no per-processor license costs.

- Maintain a repository of preapproved OSS components.
  As soon as OSS adoption reached a significant amount, you should install a central depository with approved OSS components. It is also a good idea to adjust the process to simplify the use of preapproved components. This will reduce the risk of random downloading OSS by the developers.

- Do not dwell on development processes and artifacts; focus on outcomes.
  Watch out not to destroy a time-to-market advantage OSS can provide. If you add too many too protracted activities to your process, you will also face developers who just work around the cumbersome process. Make the process as slim and dynamic as possible.

- Do not expect perfection, and plan for remediation.
  Nobody is perfect. You will need a rapid response process to handle violations. When the detected violation is no false positive, contact your legal department and evaluate the risks. After you removed the violation in all effected systems, you should forgive and re-educate an accidental violator or punish a repeating violator.

Forrester's open source Adoption Next Practices

- Join and contribute to OSS communities.
  Only few OSS adopters think about becoming committers themselves. Some take the chance to implement their needed features "on top of the core contributions of others."

- Vary support-sourcing strategies by measuring delivered value. With OSS, you have several options for support: Buy commercial support, support yourself or use a third party such as a system integrator or a specialist independent software vendor. It is necessary to analyze your needs and choose wisely from the alternatives.

- Model internal reuse strategies on the example of successful open source communities.
  Within OSS projects software reuse is quite usual in contrast to many commercial organizations. So you may use OSS as a guideline how to realize software reuse in general.

6.2 DLA Piper presentation

Mark Radcliffe, partner at DLA piper held a presentation at the open source Business Conference 2010 [2] containing the following advice: [24]

- open source is omnipresent and needs to be administrated. This process is crucial.

- The topic is cross functional and involves
  - Product planning and management
  - Legal, security and export compliance
  - Engineering

- Integrated processes for component, license and release management are required.

[2] http://www.osbc.com

- OSS rule should be integrated in the company's culture and workflow. It is activated by events such as component approval request, accepting code from a third person or performing a build.
- After a cross-functional team has composed an OSS policy, it has to be published and all affected employees have to be educated about the policy.
- There needs to be an OS process owner who "keeps the wheels running".
- During the Approval Process the component, the license and the release plan need to be reviewed and approved. The process should recognize whether the OSS is used internal, external or in products.
- The Monitoring and Tracking Process ensures that the component is verified and notifications are set up for security issues and component updates.
- The Obligation Verification Process guarantees that only approved components are used and license and business obligations are met.
- There should be criteria for approved software concerning licenses, usage (internal, product, website), sources, support and other.
- Describe under which circumstances contribution to OSS projects is allowed.
- Education of employees is necessary for compliance.
- A Compliance Core Team consists of an attorney, the OS Process Owner, a business/products agent, the technical architect and the project manager.

6.3 BigFix presentation

During his presentation at the open source Business Conference 2010 Virginia Badenhope (Associate General Counsel of BigFix, Inc.) gave the following recommendations: [2]

Goals of the legal department

- The goals of legal department are:
  - Protecting the company's intellectual property
  - IP hygiene – keeping track of all code coming in and going out
  - Decreasing litigation risk
  - Being prepared for potential M&A activity

Inventory and Assessment of OSS

- Let engineering fill a spreadsheet containing information about name, version and origin of the component as well as its license and whether it will be modified or distributed. Furthermore, it should be specified how the component is linked to commercial code and whether there is a commercial license available.
- Licenses should be categorized in risk categories.
- Note which compliance action has to be performed. For BSD license this will be just "add license information to documentation", for (L)GPL there will be more sophisticated actions.

Complying with open source Licenses

- Keep documentation up-to-date, especially all copyright and license notices and information about modifications as well as reference to source code disclosures.
- Create links on the support website to the notices mentioned above and to source code disclosures.
- Update notices in the splash screen, the user interface and the Readme file.
- Collaborate with engineers and technical writers to identify those people who are qualified to handle these matters.

Implement an open source Policy

- Prepare a policy that includes approval process, record keeping, a process for making contributions and roles and responsibilities of Legal and Engineering.
- Publish the draft to Engineering Management.
- Overhaul the draft using the comments from Engineering.
- Roll out the policy in agreement with Engineering Management.

Revise (form of) Outbound Licenses

- Include open source provision in MLA and EULA conforming to license requirements.
- Add a stipulation to reseller, distributor and OEM agreements stating that they will not modify, combine or distribute the product in a way that will subject the product under an OSS license.

Revise (form of) Inbound Licenses

- Update independent contractor agreement by adding warranty that no OSS will be included without advance disclosure and approval
- Include open source warranties stating that all open source code is disclosed in an exhibit (including name, version and information about modifications and linking) and no additional OSS is required for the product to function. The licensor also guarantees that he provides all information required to comply with the license.

6.4 IBM presentation

Bob Sutor, Vice President of open source and Linux at IBM, describes several important questions concerning open source software: [29]

Quality of OSS code

- OSS code is not "good" by itself.
- There are thousands of OSS projects and they differ significantly.
- You need a definition of "good".
- It is not assured that a project with good code right now will have good code in a month because code gets rewritten often.
- Quality of documentation and user interface may indicate quality of the code.

Impact of people and community

- Code is written and projects are driven by people. Unreliable people may provide displeasing surprises in the future.
- Work out scenarios to be prepared if the code is abandoned, forked or acquired.
- Discover whether the OSS code is developed by a community or by a company "coding in public".
- Check whether there are only developers in the community or also documenters, graphic designers and evangelists are involved.
- Get a feeling for the health of the community and take signs of trouble serious.

Legal implications

- Do not ignore legal issues! That might be the end of your business!
- Learn about licenses and think about hiring an attorney as consultant.
- If you mix open source licenses, pay attention that the mix is legal.
- Ensure that the software you want to use was developed conforming the legal rules.
- If you are not an attorney: Do not pretend to be an attorney!
- Consider, whether the license will fit to your future plans:
  - Not all licenses can be combined.
  - Not all licenses allow use of the code in closed source applications.
  - There may be restrictions hosting software as a service.
  - Pay attention when using open source libraries.
- What is your open source governance strategy?
- Understand legal risks and balance them against business value.
- Implement business and legal controls that approve use of OSS and ensure to have a well-defined escalation path.

Enterprise-readiness

- Check whether the software fits your requirements concerning security, reliability, availability, scalability, interoperability and performance.
- Check whether the software is available on the right platform.
- When you support the OSS in-house, ensure that it does not become orphaned.
- Establish a plan which updates will be installed (there may be major releases every six months) and how - if you did so - you own modifications will be propagated to new versions.
- Investigate whether the new software is easy to integrate with you preexisting software (e.g. data formats, operating system, hardware, ...)
- Specify someone how is responsible for the integration

Measurability and comparability

- Benchmarks can help examining if the software is usable for your purposes.
- Look out for head-to-head comparisons among software solutions suitable for your purpose.

6.5 OpenLogic Webinar: Six Steps to open source License Compliance

Kim Weins and Dave McLoughlin presented in their OpenLogic Webinar six actions how to get to license compliance:[35]

Create an open source Policy

The policy should consider the following aspects:

- What licenses are allowed for what kind of purpose?
- Which sources for OSS are approved?
- What is the process of approval and who makes the decision?
- In which cases is modification of OSS allowed?
- What are the conditions for support for the different kinds of purpose?
- How is Tracking and management of OSS realized?
- Do all third party suppliers comply with the policy? How to ensure that?
- What is contributed to OSS projects? How many developers of your company contribute in projects?

Define a Governance Process

- There are four types of governance processes
  - Centralized: Decisions are made by a review board or a single person.
  - Decentralized: Decisions are made by business units or groups.
  - Federated: A centralized policy can be tailored by individual business units.
  - Laissez Faire: Managers decide at will, following guidelines.

- Keep the process simple so that as many decisions as possible can be approved by the lowest management. This will sustain productivity and keep overhead low with the side effect of good scalability.
- Only some few exceptions should be escalated for more intense review. Ensure fast response, otherwise employees will find ways to go around process and policy.

Manage open source Approvals

- Start with rules that allow use of previously approved OSS.
- Install notifications every time OSS is used.
- Structure your approval process in a "standard" and a "special exception" part.
- Define criteria for approvals.
  - Your requirements: functional, viability, support, maintenance
  - License: compatible to your policy?
  - Planned use: compatible to license?
  - Risk profile
- Define responsibilities, e.g. install an OSS compliance team.
- Integrate with existing development process.
- Plan reviews and updates of the process.

Audit your Software

- Scan for used OSS projects and licenses as well as used OSS code fragments.
- Scan in server and desktop systems, applications and source code repositories.
- Scan at source code and binary level.
- Events to perform a scan:
  - OSS inserted
  - 3rd party code inserted
  - Build created
  - Before going into production
  - Product release
  - Regular schedule

- There are scanning services available as well as scanning tools (free and commercial).
- 100% OSS discovery is impossible

Analyze License Obligations

- Start with list of open source licenses
- Identify obligations for each license
- Apply usage information
- Compile consolidated list
- Identify license conflicts
- Obligations are defined by the form of usage (modification, linking, distribution)

Ensure Compliance with Obligations

- Perform final audit to confirm licenses and projects included
- Validate implementation of compliance checklist
- Perform early checks during development and a final check before releasing the product

6.6 Open Logic Webinar: Ten Key Elements of open source Governance in the Enterprise

In this OpenLogic Webinar Greg Olson and Kim Weins showed up ten elements important for open source governance:[20]

Open source strategy

- Defines reasons and goals of the organization concerning OSS
- Written in high-level business terms
- Key values of an open source strategy
  - Management consensus on the goals
  - Good foundation for evolving the open source policy
  - Good basis for future decisions

Open source policy

- Defines the conditions, how OSS may be used in the organization
- Typical elements are:
  - Legal Policy (what licenses?)
  - Acquisition Policy (criteria for OSS introduction, approval of OSS)
  - Usage Policy (Where may which OSS be used or modified?)
  - Support Policy (support requirements)
  - Management Policy (tracking, management)
  - Partner Policy (How does a 3rd party conform to the policy?)
  - Contribution and Publishing Policy (contributions by employees)

Executive sponsorship

There are challenges that can only be mastered with support from the management:

- Controversy
  - Balance between benefits and risks
  - Changes to long-established policies or processes
  - Strong opinions
- Budgetary issues
  - Needed acquisitions
  - Benefits are harder to measure than costs
- Driving the effort
  - Without a supporter from management, changes that cross management disciplines presumably will stall

Buy-in from stakeholders

- Ensures that all involved will conform to the process because a not consistently followed policy is worse than no policy
- Best ways to ensure buy-in
  - Executive leadership, especially in software development
  - All stakeholders understand the policy

  - Involve the stakeholders in the development and periodic reviews of policy and process
  - The process should produce quick approvals for frequent issues

Funding

- Provides any necessary resources
- There is no license fee, but managing the software requires resources
  - Consulting during development
  - Assessment of the code repository
  - OSS scanning, approval, tracking and management tools
  - Support
  - Indemnification

Take inventory

- Learn what OSS you use on the one hand on your servers and desktops and on the other hand in your applications
- Start inventorying concurrent to creation and implementation of OSS policy and process
- First choose a representative sample and expand bit by bit
- There are two options how to perform the inventory
  - Self reporting by surveys is inaccurate and hard to perform
  - Scanning tools can be integrated to distribution or inventory tools and find 2-10 times more than self-reporting does and can detect OSS even in binaries

Provisioning

- Use online material for research (OLEX, Ohloh, Osalt, Ostatic and project home-pages)
- Establish OSS certification (use from external supplier or create your own)
- Get your OSS from a trusted repository or pay special attention that you obtain it from an official source
- Build up an internal repository

Request and approvals

- Request and approval has to take place ahead of downloading or even use in development
- Three options to perform:
  - Manual process through e-mail which will only work in small companies
  - Tools from an external vendor that support an automated process
  - Your own automated process

Auditing

- Ensure compliance with policies ad open source licenses
- Protect your own intellectual property
- Perform at key phases (Build, Test, Staging, Production)
- Perform random spot samples
- Audit for OSS projects used, OSS licenses used and OSS plagiarism
- Compare information from policies, declarations of usage, requests and scans
- Identify and eliminate violations

Reporting

- Changes in OSS inventories
- Downloads of OSS
- Status of request and approval
- Policy compliance and violations
- Bill of materials and Bill of Licenses

6.7 Excerpt from OpenLogic Webinar: 5 Roadblocks to open source Adoption

In their Webinar "Top 10 Ways to Stretch Your Budget by Using More open source software in 2010" Kim Weins and Rod Cope discuss five possible barriers:[34]

Don't know what open source to pick

You have to filter hundreds of thousands of open source projects to find enterprise ready ones among a lot of projects that are dead, have no community, no active development, bad licenses and so on. Furthermore, there are many sources for open source, but not all are reliable and trustworthy. Define criteria to certify open source software you want to use and define which sources are allowed to obtain code from.

Concerns about support

There are three types of support:

- Community support (unreliable)
- Vendor support (not available on all projects)
- Do-it-yourself support

You should back your internal experts (if you have these) with vendor (if mission-critical) or community support.

Different acquisition process

Typically, there is a process for acquiring software facilitated by a vendor, but open source software is just being downloaded, tried and used. You should install a well-defined acquisition process also for open source software.

Concerns about risks of open source

There are some concerns about risks of open source you should address

- There is no support (there are support options available, even with SLA)
- open source means poor quality (quality varies, but many projects are equal to closed source)
- open source is not secure (open source is as secure as closed source)
- An open source project uses foreign code or patents (closed source faces exactly the same problem)

Concerns about license compliance

There are worries about license compliance:

- License proliferation: there are hundreds of licenses, some very similar
- Packages can have multiple licenses
- Packages can use other packages with other licenses
- Licenses can be incompatible to each other
- How to comply with all the terms and protect yourself from infringement?

Tips to deal with these concerns:

- Check which actions are considered distribution
- Check what additional obligations are activated by distribution
- Get "manifest" including internal and 3rd party components
- Validate that which license is relevant (look in source code!)
- Scan software (binary and/or source)
- Review project dependencies and their licenses
- Check whether there are changes made from the standard license text
- Find the obligations and conflicts of the licenses
- Search for lawsuits concerning the components
- Create "compliance checklist" for development

6.8 OpenLogic Certification Process for open source software

OpenLogic presents in this white paper the process that is performed to add a component to their OSS-repository:[23]

Open source Package Selection

- Market Acceptance
- Enterprise Coverage
- Redundancy
- Customer Requests
- OLEX Requests

Prequalification

- Initial License Review
- Source Code Availability Check

Initial Acceptance

- Viability Assessment
  - Project Structure
  - Developer Community Size
  - Committers and Contributors
  - Development Practices
  - Maturity
  - Project Lifecycle Status
  - Maintenance Activity
  - Release History
  - Project Alternatives
  - Third-Party Information
- Support Assessment
  - Community Support
  - Mailing Lists
  - Forums
  - Documentation
  - Training

  - Commercial Support
  - OXC Recruitment
- Distributions Assessment
  - Project Distributions
  - Primary Development Language
  - Stable Release
  - Commercial Version
- Legal & Security Risks Assessment
  - Legal Action/Lawsuit Search
  - Security Vulnerability Search

Full Certification and Implementation

- Detailed License Review
  - Full License Review & Validation
  - License Categorization
  - License Summary
  - Custom License Review
- Knowledgebase Development
  - Categorize Package
  - Summarize Release Notes
  - Scoring
  - Package Description
  - Key Package Information
- Packaging & Delivery
  - Verify Integrity of Distributions from Project
  - Create Standard/Custom Distributions
  - Virus Check
  - Bundled Package Analysis
  - Maven Repository
  - Discovery Fingerprint Rule
  - Secure Delivery

6.9 Navica open source Policy Process

Bernhard Golden, CEO of Navica, presents in a white paper a five phase process how to implement an open source policy:[14]

Phase 1: Evaluate open source Use Profiles

Talk to employees to understand how open source is being used in your company at the moment. When selecting your interview partners consider that people from different divisions may have different views as well as people with different roles. Investigate whether open source software is modified or distributed and if open source software may be included in software that is brought in from outside. Check on the one hand if the organization contributes to open source projects and on the other hand if employees do so in their spare time because even that may cause IP implications. Be aware that not everybody will tell the full truth, which makes a broad selection of interview partners extremely important.
After recognizing the use of open source software, you should identify the Risk Exposure, which is typically formed by license obligations as well as governmental regulations.

Phase 2: Create open source Policy

The open source Policy should address all obligations the organization has to meet as well as the process how an approval for a particular open source component is obtained accompanied by the description of the organizational resources for open source. Typically, an "open source Review Board (OSRB)" as center of expertise is vital in the process. In the OSRB, representatives from several organizations should be involved: Legal, Development, Operations, Marketing, Finance and Compliance. The OSRB approves and denies requests concerning use of open source components and gives advice how a non-approved request could be approved. The policy should also include a process of approval of contribution to an open source project.
The open source Policy should be reviewed and commented by all affected parties. Think about hiring an external lawyer if your legal department is not familiar with open source issues.
While implementing the policy you should pay attention to integrate it to existing management processes. It may be a good idea to start rollout in a pilot division with a well-established use of open source and proceed to other divisions after collecting some experience. In a decentralized company, you should plan wisely how to install the policy. One possibility is to define the policy globally and install a central "fall-back" OSRB for special cases and have OSRBs in every division that perform the day-by-day jobs.

Phase 3: Audit the Company Code Base

You need to identify all open source components you have in use to become aware of the license obligations you are subject to. You might do this by interviews or self-written tools, but neither of this would guarantee the required accuracy to you. If you want roughly complete coverage, you will get around a commercial product as offered by Black Duck Software or OpenLogic.
After identifying all open source components in use, you can place a bet that there are compliance issues with the obligations placed by the open source licenses. The next step is to quickly remove those license violations. If you used commercial software, it will provide a list with necessary actions to come to compliance.

Phase 4: Educate Employees

Education of employees should consider several aspects: An overview over open source software in general and its licenses in particular introduce the topic. The explanation in which ways the company uses open source software, a summary of the company's open source Policy and - most important - a demonstration of the relevant processes (Approval for use, Approval for contribution, Management process) will enable the employees to conform to the policy and the license obligations.
You should plan education for three different target audiences: The existing employees when installing the new policy, new employees and a periodical reminder or refresher for all employees.

Phase 5: Monitor Ongoing Policy Compliance

Once the policy is installed, the compliance to the policy needs to be monitored. The most effective way is to associate the policy's tasks to the existing project management process and to perform the open source checks along with the milestone checks. At this point, you can use the same automated methods as in Audit phase. In addition, you can integrate the monitoring also into the engineering process and perform the automated checks regularly.

6.10 Black Duck Whitepaper: The Business Case for Automating open source Code Management

In this Whitepaper Black Duck Software describes the advantages of an automated open source management process:[4]

Three Major open source Risks and Issues

- Management
  A company needs to be able to manage the open source software it uses. Someone must be capable to analyze the components in use and weigh the business and legal impact. There are examples where companies use several different components with the same functionality.
- Compliance
  While a lot of open source licenses require the source code to be published this is normally not the intention of commercial licenses. This situation leads to compliance issues that may result in court injunction against shipping a product. Another aspect is government regulation, for example concerning export of cryptographic algorithms.
- Security
  If there is no tracking of the uses open source components a company easily can miss an important security update of the component, remember Google Chrome and WebKit.

Manual Methods of managing Code are Costly and Inefficient

- Filling out forms, obtaining signatures and so on reduces developer productivity.
- An open source policy seeks to manage the major activities concerning open source:
- Search for needed components
- Select the best fitting component
- Approve the component (including legal, security, ...)
- Build and integrate the component
- Validate bay discovering license conflicts and hidden cryptography
- Monitor the usage of the component
- The *amount of code* and *number of open source components* are cost drivers when analyzing source and binary files.
- Black Duck found that analyzing code manually costs about $450 per MB.
- Initial validation of the company's source code can be extremely expensive.
- Management and Validation cost about $7800 per component and year.
- Manual methods are prune to human error.

6.11 Black Duck Whitepaper: Seven Best Practices for Managing Software Intellectual Property in an open source World

Black Duck Software suggests to follow the following practices to manage IP:[5]

Re-use existing components wherever appropriate

- When adding functionality to a project, seek internal and external developed components that could speed up development.
- Establish criteria for such components and eliminate components that do not meet those criteria.
- Eliminate every external component which license is on the prohibited license list.
- Subject the component to a make-buy analysis.

Track and control changes to internal components

- Track every creation and modification of internally developed components to protect your intellectual property.
- Establish a list with sensitive components. Count a component as sensitive if a patent is sought or the code gives you competitive advantage or something else.

Control re-use of sensitive or external components

- A project team should understand for each component they want to use:
  - The intended use and reason for use
  - The component's sensitivity
  - The way the component is incorporated
- The team should describe the technical method how the component is included (e.g. unmodified copy, static or dynamic linked and so on).
- The following four steps lead to greater control over component re-use:
  - Check whether the component was approved beforehand for the intended use.
  - Understand the license obligations of the component as well as of all other external components used by the component.

  - Understand all potential conflict between licenses of used external components.
  - Let the projects legal, technical and business authorizers approve the component.

- Approvals should be included in relevant organization-wide lists.

Verify every build and release

- Before every build or release you should ensure that:
  - No unapproved sensitive internal or external component (fragment) was added.
  - No unapproved change was made to sensitive internal component.
  - No change has been made to components where changes are not allowed.
- When detecting an incorrect change or addition, it has to be approved or removed.
- Detecting errors in early stages makes correction less expensive.
- After each build or release, the key metadata should be put in the bill-of-material.

Review compliance at project phase transitions

After a major development phase, the project legal, technical and business reviewers should:

- Verify that no unapproved components or fragments are used.
- Verify that no unapproved or prohibited changes were made.
- Ensure compliance with all license obligations.

Control component contribution and distribution

If a component is arranged to be contributed, the projects legal and business reviewers should:

- Check whether the component's sensitivity is a hindrance to contribution.
- Check whether it is legal to contribute all in the component includes external components.

Assess software components before acquisition

If you consider an acquisition that would include software components, legal, technical and business reviewers should:

- Determine all includes components not owned by the target.
- Assess the license obligations
- Assess the impact of any required rework.

7 Interviews

The interviews were performed to find out how open source adoption is handled in practice. Employees spoke or wrote about how open source is dealt with.
Comprehensive anonymity was promised to all interviewees.

7.1 Interview 1

This interview was performed over telephone with a company that develops and sells software. This company will be called Comp1.
At Comp1 the original motivations for use of open source software were reuse, cost saving and wishes by developers. These initial motives evolved to more differentiated goals. While costs savings are still relevant, there are considerations about time to market and conformity to standards. An attractive option is to make requirements by the company to standard by releasing open source components.

Comp1 also contributes to open source projects because they realized that there are several advantages when doing so: One the one hand via giving back the bug fixes to the community in future the integration will be less elaborate; on the other hand such contributions keep the community and innovation alive. Furthermore, open source can be a way of investment protection: When a former proprietary component is released as open source and widely adopted, it can evolve to a new standard.

With all these opportunities in mind, an evaluation is performed whether the use of open source is reasonable, whether the particular component fulfills the requirements by the company and if the use could cause additional effort during integration.
The open source process at Comp1 is organized as follows:

- Any developer who wants to use an open source component, declares the need, accompanied with a concrete description, in which way the component will be useful
- Several aspects of the component will be documented, like the license, the activity of the community, whether other companies use the component as well and the result of an intellectual property analysis
- A decision is made, separate for each use case and release, based on the experience the component and the cost for a potential replacement of the component.

- An automated check ensures that there are no unapproved components in use

Several problems arise in this context: Some employees might not know about the process, others do not like it or had bad experiences and ignore it therefore. In some cases the process may be too slow, decisions may take too long, which seems to drive developers to work around the process. This is avoided by designing the process fast and efficient, by delegating the responsibility and by naming concrete contact people. Tools are used to identify all open source components Comp1 works with and the resulting list is used to speed up approval and to ensure that there are not many components with the same functionality. An important aspect is scalability because there is a goal to increase use of open source significantly. An executive sponsor eases the handling notable.

One big challenge are the employees. It takes years to raise the awareness for open source software and its implications. Objective is to get the understanding that open source is a possibility to become better and faster, but it incorporates risks. Many developers do not know about the risks and implications attached to open source. Generally, developers and attorneys do not speak the same language, which makes it necessary to have an instance that acts as link between the two worlds.

7.2 Interview 2

This was a written interview with a company that develops and sells software. This company will be called Comp2.
Comp2 uses open source software to reduce costs, reduce time to market and to avoid development of commodity components.

Open source components are classified, comparable to Basel I-III concerning the finance sector. Dependent from the classification a component may be used without or after management approval. Viral licenses are generally excluded from classification and have to be assessed in-depth. Components licensed under other licenses are classified after an initial intensive review process, including patent research. Attorneys specialized on intellectual property approve a component, but management bears responsibility.
Comp2 protects itself from releasing code as open source, caused by viral effect of a license and contamination by accidentally and unconscious insertion of licensed code. Another problem are patent license grants that may be given when contributing to an open source project under a specific license.

Problems arise because IP-attorneys focus on approval and the needs to be built up expertise in development departments concerning open source and intellectual property. For complete understanding of the situation, process knowledge is necessary, but questions about process are mainly handled by project management, who have sparely technical knowledge and connection to development teams is rather unsatisfactory.

7.3 Interview 3

This interview was performed over telephone with a company that develops and sells software. This company will be called Comp3.
Comp3 traditionally separated external from internal code. They introduced an open source process as a customer asked for a bill of material containing all used open source software. They installed a central "open source Clearing House" that acts as contact for inquiries and approves requested open source components. Representatives from CTO, Procurement and Legal are involved in this board. The individual business units choose on their own how they forward components to the Clearing House. For example, developers can start a request for a particular component, giving information about the component like license, source and whether it is a new component or just an update. Architects check, whether there is already a component with the same or similar functionality approved. Finally, the component is handed to the Clearing House. After approval, the component is included in the business unit's repository. This repository contains the approved components accompanied with notes whether the approval is general or just for a specific project. Open source software is not used in core business.

An automatically performed Black Duck scan ensures compliance to the process. This scan is not very popular among the developers because they do not like their code to be inspected. With trainings, awareness for the possible problems should be created. While the trainings are well attended and the employees seem to recognize the sensitivity of open source, they still do not like their code to be inspected.

Comp3 also contributes back to open source projects; official objective is to contribute back all bug fixes. If there has been further development, contribution has to run through a review process to avoid loss of company intellectual property.

A problem arises if approval in a particular case takes abnormally long because there are sophisticated legal issues to resolve. In this case, and other as well, developers get very creative how to get the desired code into the project without waiting for an approval. If successful, there is the possibility that the open source code remains undetected until the next delivery, which may be month later. A legal problem[1] is the use of code that is published "to the public domain". In German copyright, the term "public domain" does not exist. Therefore, you need to come to an agreement with the author who often does not like inquiries because he "released the code to public domain". Other authors build up their own licenses by putting parts from different licenses together. This always requires extensive research by legal department to identify possible obligations and incompatibilities.

[1] This problem exists in Germany. Other countries may not be affected.

7.4 Interview 4

This was a written interview with a company that counsels other companies in open source topics and offers tools to handle open source software. This company will be called Comp4. Comp4 mentions three aspects to consider: legal (adjustment of contracts, risks, elucidation), process and tools (automatic scans, integration in existing ticket systems, local repositories). A process should consider:

- Initial scan for open source code to determine status quo
- Base lining leading to approved bill of materials
- Review of open source software
- White- and Black-Lists
- Selection process for developers
- Approval process
- open source team

Benefits provided by open source software can be:

- Cost savings
- Increased quality
- Concentration on core business, no development of commodity components
- Developers learn from practices uses developing open source software

Possible risks that should be considered are:

- Ignorance, what is contained in the code base
- Loss of intellectual property through contribution
- Employees bypassing the process
- Injunction
- Negative public relations

Comp4 considers the employees as the biggest challenge on the way to an open source process.

7.5 Interview 5

This interview was performed over telephone with a company that develops and sells software. This company will be called Comp5. There is a group at Comp5 that is responsible for all matters of foreign IP. When a developer wants to use code he has not written himself (even it's the code from his colleague), he has to put a request for that code. The group analyzes the request, arranges things with legal department and gives instructions to the developer how to proceed. In doubt, the issue is discussed together with the developer and his boss(es). Every approved component is added to a repository, which serves as a white list for the developers: any component on it may be used without further authorization. For any new version of a "white-listed" component, a new request is required, because every new version could be released under another license. The main criterion for approval of an open source component is the license under which it is released. There are predefined responses for the established licenses, depending from the use case scenario (internal use or external distribution). As the company's development model stipulates suite releases[2], the high workload in the group becomes a bottleneck up to two month before such a suite release. Several aspects allow an effective and efficient implementation of the process:

- The whole process is supported by a self-developed tool
- An Automated code scan checks for unapproved (accidentally) incorporated open source code
- Another scan is performed to detect security vulnerabilities
- The top management provides good executive sponsorship by declaring the topic a corporate risk

The company performs detailed trainings whenever new employees are hired or another firm is acquired. Employees learn about the policy and process as well as the backgrounds. The process is non-restrictive, every request refusal is justified and discussion is encouraged, which leads together with the trainings to a high adherence to the process. There are no attempts to circumvent the process. Comp5 makes well-targeted contributions to open source projects to influence the development and to stay well informed. Condition for any contribution is that the project is managed by a foundation. Comp5 assesses any firm they acquire and if the risk seems to be too high, the acquisition process is even canceled. As risks, source code disclosure and patent issues are named.

[2] all products of a product line are released coincidentally

8 Business Risks

This chapter describes the most important threat scenarios that were identified in the examined literature and performed interviews. Where possible, less frequent mentioned aspects have been integrated into the more important. The described scenarios are not disjunct, some may entail other, but these can in turn occur independently.

We also created a model that structures the risks and puts them in relation to each other. A survey was launched to verify this model. Becuse by the deadline of this thesis there were not enough results available, the model and the survey were put in appendix. For each threat scenarios the findings on which it is based are given. Findings in related works are referenced by their number in bibliography, findings in interviews by timecode (mm:ss).

8.1 Infiltration with open source code

There two main ways how open source code gets into your company unseen: One the one hand any third party component you use might use open source code itself without declaration. On the other hand, can bring in open source code accidentally (mainly) and consciously (regrettably, but true). Imagine the following examples: A developer reads a book or uses an open source component at home. Months or years later, he remembers a special algorithm or code structure and uses it in office. Another developer wants to use a specific open source component, but does not get an approval for it (or had the experience that approvals take far too long); he uses the component anyway and tries to hide the usage by linking only a binary[1].

References: Interview 1 (24:58): OSS brought in by developers, Interview 2: contamination, Interview 3 (17:02): OSS brought in by developers, Interview 4: OSS brought in by vendors, Interview 5 (31:45): OSS brought in by developers, [2, 4, 15, 35]

8.2 Potential mergers and acquisitions

If your company is to be acquired its value is dependent of the accuracy and completeness of your recordings concerning open source usage.

[1]Such case happened in reality and was told in an interview

References: Interview 5 (11:42): potential M&A, [2, 24, 35]

8.3 Customers ask for bill of material

There will be customers that will ask you for a bill of material. This will be in doubt a knockout criterion.

References: Interview 3 (02:05): customers ask for BoM, [24, 35]

8.4 Lawsuit

The apparently most formidable threat is a possible lawsuit. Let the result be just an injunction, a stop of delivery or even a recall campaign, the impact can be disastrous.

References: Interview 1 (17:35): lawsuit, Interview 3 (40:31): lawsuit, Interview 4: lawsuit, [2, 4, 15, 24, 35]

8.5 Loss of intellectual property

The hazard of loosing intellectual property exists mainly when contributing to open source projects. As some open source licenses grant patent rights you should be sensible when contributing in areas where you hold patents. A more general danger is that someone could draw conclusions by inspecting the open source code you released.

References: Interview 2: loss of IP\patent license grant, Interview 3 (38:47): loss of IP, Interview 4: loss of IP\patent license grant, Interview 5 (22:54): loss of IP, [2, 24, 35]

8.6 Negative PR

In case of inadequate usage of open source, the issue can get into media focus and there may arise a public discussion.

References: Interview 4: negative PR, [4, 14]

8.7 Infringement of Intellectual Property

There is a variety of possible infringements of intellectual property. You might infringe in a direct manner by using code without the right to do so. There can also be a rather indirect infringement when you use a component that infringes someone's intellectual property.

References: Interview 1 (19:02): IP infringement, Interview 2: IP infringement, [35]

8.8 Monetary damage

Monetary damage can be caused by various reasons. Maybe you are forced to replace a component by another, implying additional financial effort or you may be compelled to pay statutory damages.

References: Interview 3 (40:31): monetary damage, [4, 24]

8.9 Release Code as open source

Many open source licenses require releasing code as open source if it is combined in a specific manner (different from license to license) with code that is licensed under that license.

References: Interview 2: release code as open, Interview 3 (48:23): release code as open, Interview 5 (21:08): release code as open, [15]

9 Best Practices

This chapter describes the most important practices that were identified in the examined literature and performed interviews. Where possible, less frequent mentioned aspects have been integrated into the more important. With these practices, companies can respond to the threat scenarios described in chapter 8.
The determined categories were associated with one of three groups:

Policy (9.1) These practices affect the open source policy (-paper) itself.

Process (9.2) These practices induce the daily used open source Process.

Infrastructure (9.3) These practices influence the environment in which open source is used including staff and tools.

In this chapter, the most mentioned practices are described in form of patterns as follows:

Context The context in which the pattern is to be used. Some patterns do not have a specific context because they are initial and should be performed anyway.

Problem The description of the problem, arising in the given context (if any) or permanent present[1]

Solution Advice how to handle the problem, perhaps to solve it.

References Findings that led to this practice. Findings in related works are referenced by their number in bibliography, findings in interviews by timecode (mm:ss).

9.1 Policy

9.1.1 Understand licenses

Context Creation of an open source Policy

Problem There are uncountable open source licenses out there. Even the open source Initiative (OSI) has approved more than 60 licenses. Some licenses are rather complex to understand and some authors mix existing licenses at their will.

[1] Someone might think there is no open source related problem if you don't use open source software. Of course that might be correct, but how to prove that no open source software is used?

Solution In order to provide a correct license classification it is necessary to comprehend all implications an open source license has. At this point counsel of a lawyer can prove helpful because not all obligations may be obvious to a legal nonprofessional. Self-written or -mixed licenses may contain discrepancies that make a legal use of the open source software impossible.

References Interview 5 (13:35): understand licenses, [5, 14, 15, 23, 29, 35]

9.1.2 Classify by license

Context Creation of an open source Policy

Problem Alone SourceForge hosts more than 260,000 open source projects and there are many more. [28] This is an extreme variety difficult to overlook.

Solution On the other hand, Black Duck Software states that more than 93% of the open source projects listed in their database use a license that is under the Top 10 open source licenses and nearly 97% use a license under the Top 20. [3] Therefore, you can reduce complexity by four orders of magnitude by classifying by the license that is used.

References Interview 1 (16:48): classify/criteria\license, Interview 2: classify/criteria\license, Interview 3 (23:08): classify/criteria\license, Interview 5 (09:25): classify/criteria\license, [2, 5, 15, 20, 23, 24, 34, 35]

9.1.3 Classify by type of usage

Context Creation of an open source Policy

Problem There is no "usage" of open source. There are typically up to four types of "usage" of open source software: normal usage as an application (e.g. download, install and use Open Office), modification, distribution (again differentiated whether modified or original code is distributed) and "making available" (e.g. Wiki-software that is hosted on a server)

Solution Consider under which circumstances you will "use" the open source software. Also, think about the future, even perhaps far future. Might it be that the now hosted software wins an award in five years and you want to offer some good customers the possibility to host it on their own site? That would be distribution. Especially when using open source licensed under GPL this consideration is very relevant because the GPL allows hosted applications without reveal of source code.

References Interview 5 (49:14): classify/criteria\usage, [5, 14, 15, 20, 24, 29, 35]

9.1.4 Evaluate support options

Context Creation of an open source Policy

Problem Since there is not a vendor where you buy the component like proprietary software, the question about support arises.

Solution In general, there are three options for support: Community support, Vendor support (there are companies offering support for specific open source components) or self-support.

References [20, 23, 24, 34, 35]

9.1.5 Keep bill of material up to date

Context Creation of an open source Policy

Problem Someone might ask you which open source software you use. This may be your contractual partner, but also someone else might allege that you use a specific open source component.

Solution To answer such requests or repulse false allegations you should always know what is in your code base. Therefore, you need to build up a bill of material containing every single component that is used in a project. For a reliable way to get a complete bill of material and keep it up-to-date, see 9.3.4 and 9.3.7.

References Interview 3 (03:37): BoM/BoL, Interview 4: BoM/BoL, [2, 4, 20, 34, 35]

9.1.6 Define conditions for contribution

Context Creation of an open source Policy

Problem At some point the wish to contribute back will arise, be it a political decision, the practical consideration to minimize integration effort or just an individual ambition of a developer. This puts some new risks, for example may secrets be published or patent licenses granted[2], see also chapter 8.

Solution Consider all possible risks contribution can hold. Decide for each risk how severe it is and define rules.

References Interview 3 (36:37), Interview 5 (24:34), [2, 5, 14, 20, 24, 35]

[2] some open source licenses contain a patent license grant

9.1.7 Educate

Context An open source policy has been created

Problem Paper does not blush. Software developers are usually no intellectual property specialists and might not comprehend the full magnitude of consequences open source adoption can have.

Solution After creation of an open source policy, it is necessary to roll it out and train all involves employees. All stakeholders need to get an understanding of implications caused by open source software and how the rules in the open source policy handle these problems. They need to realize that circumventing the policy or the process can cause fatal damages to the company. As whole, it might be helpful to present the issue in a positive way, not focused on risk, restrictions and additional effort, but on chances and support to use the chances without trouble.

References Interview 1 (26:40), Interview 3 (13:01), Interview 4, Interview 5 (33:35), [2, 14, 15, 20, 24]

9.2 Process

9.2.1 Approval

Context An open source component is intended to be used

Problem Someone wants to use a specific open source component in a product.

Solution There needs to be a decision whether the desired component may be used in that context or not. This decision should be based on the open source policy and evaluate the criteria defined there. It should be clear who is allowed to make this decision. It should not take too long until the decision is made because a developer who has to wait long for such a decision will think twice whether he will put such a request next time.

References Interview 1 (17:55),Interview 2,Interview 3 (04:59),Interview 4, Interview 5 (41:05), [2, 4, 5, 14, 20, 24, 29, 34, 35]

9.2.2 Check for incorporated legal issues

Context An open source component is intended to be used

Problem No author guarantees that an open source component is “legally clean”

Solution You should check every open source component whether there is any lawsuit or whether there are license incompatibilities, e.g. a component my use another with a different license and may not comply fully with that license. The desired component might use other components that are licensed under a license you do not like.

References Interview 1 (17:15), Interview 2, Interview 3 (42:15), [4, 5, 29, 34]

9.2.3 Learn about the community

Context An open source component is intended to be used

Problem The community has a big influence on the quality of the developed component

Solution Before an open source component is used, you should do some research about the community. Interesting questions may be: How many developers work on the project? How many people use the component? When was the first release? How often are releases published? How is the community organized? Is there a company or foundation involved?

References Interview 1 (22:05), Interview 2, [23, 29, 34]

9.2.4 Get warranties from contractors

Context External software is acquired

Problem You do not know what software a contractor integrated in the software he offers to you

Solution You need to get from every contractor a complete bill of material stating which third party software is included in his software. Additional the contractor should guarantee that he listed all used software that you can claim compensation if it turns out that a not declared open source component had been used whose license forces you to any unpleasant action.

References Interview 4, [2, 5, 20, 35]

9.2.5 Verify obligations

Context An open source component is in use

Problem After approval of an open source component, someone might breach the obligations

Solution To ensure license compatibility, it is necessary to verify whether all obligations that are set up by the license are met.

References Interview 1 (27:27), Interview 5 (40:10), [5, 20, 24, 34, 35]

9.3 Infrastructure

9.3.1 Collaborate cross-functional

Context Implementing an open source process

Problem The aspects that have to be considered along with open source usage are not classic IT topics

Solution Involve people from different departments like Development, Security, Quality, Legal, Procurement, ... It is important not to disregard a discipline because sooner or later there may occur problems. E.g. if legal aspects were neglected you might face a suit or if business aspects are overlooked it might put the companies health at risk.

References Interview 2, Interview 3 (05:18), Interview 5 (05:03), [4, 14, 24]

9.3.2 Install a compliance core team

Context Implementing an open source process

Problem You need someone who is responsible for these issues

Solution Install a cross-functional team (see 9.3.1) that serves as contact for anyone who has questions about open source issues. It is a good idea to let this team also handle open source requests and approve open source components.

References Interview 1 (27:00), Interview 3 (04:23), Interview 4, Interview 5 (05:03), [14, 20, 24, 35]

9.3.3 Integrate in existing processes

Context Implementing an open source process

Problem The best process and policy are worthless if they are not followed

Solution The best way to ensure that process and policy are actively lived is to integrate them into the existing processes. For example, the Acquisition process may be expanded to include instructions for open source components. At milestones during the development process a complete check for open source may be performed.

References Interview 1 (28:09), Interview 3 (11:20), Interview 4, [4, 5, 14, 20, 24, 35]

9.3.4 Use code scanners

Context Implementing an open source process

Problem Manual search for open source in the code base is expensive and always incomplete

Solution Code-scanning tools as offered by companies like Black Duck, Palamida or OpenLogic facilitate an effective and efficient analysis of your code base. These tools are able to scan binary files, which is rather impossible to do manually. Using those scanners will ensure that you have a list of open source software that is as complete as possible.

References Interview 1 (27:35), Interview 3 (11:03), Interview 4, Interview 5 (39:28), [4, 14, 15, 20, 34, 35]

9.3.5 Maintain a repository of pre-approved OSS

Context Implementing an open source process

Problem Many open source components will be used more than once

Solution Every approval of open source takes time and thereby costs money. Install a repository with open source components that may be used without further approval. You could distinguish between different cases of usage (see 9.1.3). As soon as noteworthy number of components is included in this repository it will significantly speed up your process.

References Interview 1 (31:33), Interview 2, Interview 3 (06:13), Interview 5 (10:02) Interview 4, [5, 15, 24, 35]

9.3.6 Establish a monitoring and tracking process

Context Implementing an open source process

Problem Open source and its usage is not static

Solution You should establish a tracking and monitoring process that covers two aspects: One the one hand the usage of a specific open source component in your company may change. Probably a component licensed under GPL and approved for hosting is (accidentally) used in a product you will distribute. On the other hand, development of the open source component itself has to be tracked. You

should stay up-to-date which security issues are known about a specific component and you have to decide which new versions[3] you want to adapt.

References [4, 14, 20, 24, 35]

9.3.7 Create an open source inventory

Context *any*

Problem Initially you do not know which open source software you use

Solution For every company it is important to know which open source software they use. Therefore, you should build up an open source inventory containing all used open source components. Code scanners (see 9.3.4) provide a reliable way to do so.

References Interview 1 (30:09), Interview 4, Interview 5 (09:55), [2, 5, 14, 20]

[3] Pay attentions with new versions: every new version could be licensed under another license.

10 Conclusion

This thesis showed that open source software in general is suitable for commercial use. This does not apply to every license and every project, but overall open source software can be used in a closed source context.
The number of practices that are to consider is manageable. If the suggested practices are performed meticulously and consistently, they provide a good way to get a handle on open source software.
The crucial point are the employees. As usual in software development, a lot depends on the developers. If they are aware of the potentially critical business risks, they will ease the process a lot. If they are unaware or just do not care about the risks the company may be exposed to, they can become very creative to work around the process and it would become difficult to keep on compliance.
As usual, patents are and stay a difficult topic and equal difficult to handle.

Limits to Study

This thesis covered software developing companies, others that are also affected by open source software, but do not develop, were not regarded.
The interviews were performed in an explorative, qualitative style. Therefore, there was no strong scientific method used for creation of the questions and evaluation.

Recommendations

Very important is how you see and communicate open source software. It should be seen as big chance with some risks (motto "no risk no fun") not as a threat by itself.
Developers should establish something like a global responsibility to care for business issues that do not directly affect their daily work.
The open source process should become as a matter of course as many other processes are already.
Every organization developing software should work with a code scanner to get a comprehensive assessment of the open source software in use.
Aside all advantages and chances of open source software do not underestimate the impact of intellectual property. It is wise to have an attorney specialized on these topics at your side.

When you use open source, contribute back! There are two reasons why to do so: On the one hand it is just fair because you used the code for free. On the other hand, it is a sort of investment protection when you contribute your bug fixes and keep the community vivid.

Further research

Obviously the evaluation of the survey on business risks is still to be done. With these results the introduces model is to be revised.

Similar research can be done on the suggested best practices. A possibility is to develop a framework of best practices that may guide companies to open source compliance.

There are two aspects concerning metrics that deserve further attention: First, metrics to measure – or even better to predict – the community would be very useful. Second, there seem to be no metrics that allow explicit measurement of final cost savings by open source software.

A different aspect are the circumstances under which companies contribute to open source projects.

The context of research can be extended to other countries and it can be performed in a more standardized quantitative manner.

A Codings

see next page...

	Business Risks	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
R1	business continuity		•														
R2	cloudy IP situation in OS component		•														
R3	confusing variety of OSS						•										
R4	governmental regulations						•										
R5	security						•										
R6	customers ask for BOM			•						•						•	
R7	IP infringement	•	•													•	
R8	negative PR				•		•						•				
R9	potential M&A					•			•	•						•	
R10	release code as open		•	•		•					•						
R11	lawsuit	•		•	•		•		•	•	•					•	
R12	injunction				•					•							
R13	monetary damage			•			•			•							
R14	indemnity									•							
R15	loss of IP			•	•	•			•							•	
R16	patent license grant		•		•	•											
R17	contributions by employees									•							
R18	contamination		•														
R19	OSS brought in by ...								•							•	
R20	developers	•		•	•	•	•				•					•	

continued on next page

	Business Risks *(continued)*	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
R21	system integrators										•					•	
R22	vendors				•						•					•	

Practices	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
infrastructure																
processes																
integrate in existing processes	•		•	•		•	•		•			•			•	•
monitoring & tracking						•			•			•			•	•
violation response process							•			•						
project leaders identify OSS dependencies										•			•			
architects regulate OSS exploitation and maintenance			•							•						
tools				•												
repository of preapproved OSS	•	•	•	•	•		•		•	•					•	
use code-scanners	•		•	•	•	•				•		•	•		•	•
open source inventory	•			•	•		•	•				•				•
staff																
adjust employee contracts				•	•											
knowledge about OSS and IP		•														
executive sponsoship	•				•										•	•
compliance core team	•		•	•	•				•			•			•	•
cross-functional collaboration		•	•		•	•			•			•				

continued on next page

Practices *(continued)*	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
process				•						•						•
actions																
evaluate all types of software										•						
calculate costs for replacement	•															
obligation verification	•				•		•		•				•		•	•
perform compliance action			•					•								
approval	•	•		•	•	•	•	•	•			•	•		•	•
per project/usecase	•		•													
license review			•				•		•				•		•	
usage (int, ext, prod)									•							
component review									•		•					
ilities - realistic assessment										•						
learn about the community	•	•									•		•	•		
consider long-term consequences										•	•					
get warranties from independent contractors				•			•	•							•	•
check for legal issues incorporated in the component	•	•	•			•	•				•		•	•	•	
infrastructure																
use OSS for better deals with commercial suppliers										•						
start with widely adopted applications										•				•		
fast approval for mainstream issues	•		•	•											•	•

continued on next page

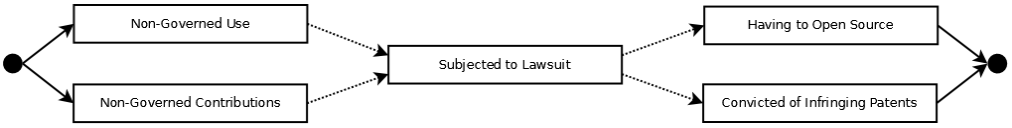
Practices *(continued)*	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
front-load evaluation and selection							•			•			•			
focus on process outcomes										•						
purchase authority is no effective process control										•						
scalability	•		•		•							•			•	
policy								•	•	•						
creation																
goals and aspirations										•	•					•
understand licenses					•		•			•	•	•		•	•	
involve lawyer	•			•						•	•					
establish in organization/educate	•		•	•	•			•	•	•		•				•
circumstances																
social aspects	•		•						•							
no once-and-done task										•					•	•
meet the needs								•		•		•				•
KISS									•	•						
contents																
conditions for contribution			•		•		•	•	•			•			•	•
describe compliance action					•			•				•	•		•	
handle external and sensitive internal components						•	•									
scope of coverage								•	•							

continued on next page

Practices *(continued)*	Interview 1	Interview 2	Interview 3	Interview 4	Interview 5	Automating [4]	Managing IP [5]	BigFix [2]	DLA Piper [24]	Forrester [15]	IBM [29]	Navica [14]	Roadblocks [34]	CertificationProcess [23]	License Compliance [35]	OS Goverance [20]
roles and responsibilities							•	•				•			•	
BoM/BoL			•	•		•		•					•		•	•
classify/criteria		•	•	•		•	•						•		•	•
usage by other companies	•															
community	•															
sources									•				•		•	•
support									•				•	•	•	•
license	•	•	•		•		•	•	•	•			•	•	•	•
usage					•		•		•	•	•	•			•	•

B Business Risks Model

Starting with the insights from interviews and related works, we restructured the identified risks, defined groups and evolved the following model. Within this model we describe cause-effect relations, while the cuases themselve are dependent from each other:



We started a survey to validate this model, but at the moment of the deadline for this thesis there were not enough responses to draw well-founded conclusions.

see next page...

Cause		Effect		Explanation
1	Non-governed use of open source in your software products	**1.1**	Can't sell to customers whose bill-of-material expectations are not met	Some customers require a bill-of-materials and won't buy if components don't meet their requirements. Not knowing what's in your company's products can lead to missed sales.
		1.2	Lowered software or firm acquisition price	If your software or company is being acquired, the acquiring firm typically checks for open source components and possible problems. If undesired open source components are found the attainable price may be lower.
		1.3	Increased risk of being subjected to lawsuit	Non-governed use of open source may increase the risk of being subjected to a lawsuit.
2	Non-governed contributions to open source projects	**2.1**	Loss of intellectual property	Through non-governed contribution, an employee might provide an unwanted license grant to the firm's intellectual property, for example, a patent license grant, to the open source project.
		2.2	Unintentional product strategy revealing	When your company contributes to an open source project (some fix or adaption) third parties may be able to learn about your product strategy by inspecting the code you contributed.
		2.3	Unintentional product structure revealing	When your company contributes to an open source project (some fix or adaption) third parties may be able to learn about the internals of your product by inspecting the code you contributed.

continued on next page

Cause		Effect		Explanation
3	Subjected to lawsuit	**3.1**	Bad press (as lawsuit defendant)	Bad press and lower public opinion may lead to missed sales and a variety of other problems.
		3.2	Legal costs	A lawsuits leads to direct legal fees and other financial costs that may not be recuperated later.
		3.3	Diverted attention	Lawsuits cost time that is taken away from taking care of regular business. This may hurt running your business effectively.
		3.4	Unwanted revealing of business information	Legal proceedings may force you to reveal business information that you'd rather keep secret from your competition and the general public.
		3.5	Contingency planning costs	Once involved in a lawsuit, you need to prepare for the event of loosing it. Such preparation, in particular added accounting liabilities, may hurt your business.
		3.6	Settlement costs	A settlement out of court comes with additional financial costs as well as potential software rework costs or license fees.
		3.7	Lost lawsuit	A lost lawsuit may force you to open source, pay damages, etc.

continued on next page

Cause		Effect		Explanation
4	Having to open source (because of lost lawsuit)	**4.1**	Bad press (for having withheld open source)	Bad press and lower public opinion may lead to missed sales and a variety of other problems.
		4.2	Supporting competitors	Code you open source may benefit your competitors who can now utilize your work for free, depending on their business model.
		4.3	Loss of intellectual property	Depending on the applicable license, open sourcing may result in a patent license grant. You may lose exclusive usage rights.
		4.4	Unwanted revealing of product strategy	Competitors may learn about your product strategy from investigating your open source contributions.
		4.5	Increased law suit vulnerability	The increased transparency of (some of) your source code makes it easier for competitors and third parties to find reasons to sue you.
5	Convicted of Infringing Patents (in Lawsuit)	**5.1**	Bad press (for infringing patents)	Bad press and lower public opinion may lead to missed sales and a variety of other problems.
		5.2	Having to pay damages	You may have to pay potentially high damages to the claimant for infringing their patents.
		5.3	Rework or license fees	You either incur rework costs or patent license fees. Given the situation of a lost lawsuit, these costs may be high.

Bibliography

[1] The Apache Foundation. Apache License: Version 2.0, 2004.
URL http://www.apache.org/licenses/LICENSE-2.0

[2] Virginia Badenhope. Pulling It All Together: An Open Source Action Plan for In–House Counsel, 18.03.2010.
URL http://www.osbc.com/

[3] Black Duck Software. Open Source License Data, 07.10.2010.
URL http://www.blackducksoftware.com/oss/licenses#top20

[4] Black Duck Software. The Business Case for Automating Open Source Code Management, 2009.
URL http://www.blackducksoftware.com/resources/whitepapers

[5] Black Duck Software. Seven Best Practices for Managing Software Intellectual Property in an Open Source World, 2009.
URL http://www.blackducksoftware.com/resources/whitepapers

[6] Rouven Buchtala. *Determinanten der Open Source Software-Lizenzwahl: Eine spieltheoretische Analyse: Univ., Diss.–Freiburg (Breisgau), 2007.*, volume Bd. 12 of *Informationsmanagement und strategische Unternehmensführung*. Lang, Frankfurt am Main, 2007. ISBN e9783631571149.
URL http://www.gbv.de/dms/ilmenau/toc/546012639.PDF

[7] Thomas Dreier et al. *Software- und Computerrecht*. Verl. Recht und Wirtschaft, Frankfurt a.M., 2008. ISBN 9783825229382.
URL http://www.gbv.de/dms/ilmenau/toc/533184770.PDF

[8] The Eclipse Foundation. Ecipse Public License (EPL) Frequently Asked Questions.
URL http://www.eclipse.org/legal/eplfaq.php

[9] The Eclipse Foundation. Eclipse Public License.
URL http://www.eclipse.org/org/documents/epl-v10.php

[10] Free Software Foundation. GNU General Public License: Version 3, 2007.
URL http://www.gnu.org/licenses/gpl.html

[11] Free Software Foundation. GNU Lesser General Public License: Version 3, 2007.
URL http://www.gnu.org/licenses/lgpl.html

[12] GBdirect Ltd. Benefits of Using Open Source Software.
URL http://open-source.gbdirect.co.uk/migration/benefit.html

[13] Barney G Glaser et al. *Grounded Theory: Strategien qualitativer Forschung*. Programmbereich Gesundheit. Huber, Bern, 3., unveränd. aufl. edition, 2010. ISBN 9783456849065.

[14] Bernhard Golden. Creating and Implementing An Open Source Policy: Managing Use and Reducing Risk.
URL `http://www.blackducksoftware.com/resources/whitepapers`

[15] Jeffrey S. Hammond. *Best Practices: Improve Development Effectiveness Through Strategic Adoption Of Open Source.* Forrester, 2009.

[16] Till Jaeger et al. *Open Source Software: Rechtliche Rahmenbedingungen der Freien Software.* Beck, München, 2. aufl. edition, 2006. ISBN 3406538037.
URL `http://www.gbv.de/dms/spk/sbb/recht/toc/490818854.pdf`

[17] Philipp Mayring. *Qualitative Inhaltsanalyse: Grundlagen und Techniken.* Beltz Pädagogik. Beltz, Weinheim, 11., aktualisierte und überarb. aufl. edition, 2010. ISBN 9783407255334.

[18] Mozilla Foundation. Mozilla Public License.
URL `http://www.mozilla.org/MPL/MPL-1.1.html`

[19] Netcraft. December 2010 Web Server Survey, 2010.
URL `http://news.netcraft.com/archives/category/web-server-survey/`

[20] Greg Olson and Kim Weins. Ten Key Elements of Open Source Governance in the Enterprise, 17.06.2009.
URL `http://www.openlogic.com/downloads/webinars.php`

[21] Open Source Initiative. The Open Source Definition.
URL `http://www.opensource.org/docs/osd`

[22] OpenBRR.org. Business Readiness Rating for Open Source, 2005.
URL `http://www.openbrr.org`

[23] OpenLogic. OpenLogic Certification Process for Open Source Software.
URL `http://www.openlogic.com/downloads/whitepapers.php`

[24] Mark Radcliffe. Managing Open Source Software 2010: Best Practices, 17.03.2010.
URL `http://www.osbc.com/`

[25] Thomas Renner. *Open Source Software: Einsatzpotenziale und Wirtschaftlichkeit; eine Studie der Fraunhofer-Gesellschaft.* IRB-Verl. and Fraunhofer-Inst. für Arbeitswirtschaft u. Organisation, Stuttgart, 2005. ISBN 3816770088.
URL `http://www.gbv.de/dms/bsz/toc/bsz252427572inh.pdf`

[26] Dirk Riehle. Control Points and Steering Mechanisms in Open Source Software Projects.
URL `http://dirkriehle.com/publications/2010/control-points-and-steering-mechanisms-in-open-source-software-projects/`

[27] Manuel Sojer et al. Ethical Considerations in Internet Code Reuse: A Model and Empirical Test. May 2010.

[28] SourceForge. About.
URL `http://sourceforge.net/about`

[29] Bob Sutor. Asking the Hard Questions about Open Source Software, 17.03.2010.
URL `http://www.osbc.com/`

[30] United States Patent and Trademark Office. What Are Patents, Trademarks, Servicemarks, and Copyrights?
URL `http://www.uspto.gov/web/offices/pac/doc/general/whatis.htm`

[31] University of California. The BSD License.
URL `http://www.opensource.org/licenses/bsd-license.php`

[32] Vercacode. State of Software Security Report Volume 1, 2010.
URL `http://www.veracode.com/images/pdf/executive_summary_veracode_state_of_software_security_report_volume_1.pdf`

[33] Kim Weins. Open Source ROI: Uncovering the Hidden Costs, 28.03.2007.
URL `http://www.openlogic.com/downloads/webinars.php`

[34] Kim Weins and Rod Cope. Top 10 Ways to Stretch Your Budget by Using More Open Source Software in 2010, 09.12.2009.
URL `http://www.openlogic.com/downloads/webinars.php`

[35] Kim Weins and Dave McLoughlin. Six Steps to Open Source License Compliance for ISVs, 05.08.2009.
URL `http://www.openlogic.com/downloads/webinars.php`

[36] David A. Wheeler. Secure Programming for Linux and Unix HOWTO, 1999.
URL `http://www.dwheeler.com/secure-programs/`

[37] James A. J. Wilson. Benefits of open source code, 06.09.2010.
URL `http://www.oss-watch.ac.uk/resources/whoneedssource.xml`

[38] World Intellectual Property Organization. Berne Convention for the Protection of Literary and Artistic Works.
URL `http://www.wipo.int/treaties/en/ip/berne/trtdocs_wo001.html`