UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

License Usage Analysis and License Recommendation in Open Source Software Development

Masterarbeit

zur Erlangung des Grades Master of Science
im Studiengang Informatik

vorgelegt von
Kevin Schmidt

Erstgutachter:	Ralf Lämmel Institut für Informatik
Zweitgutachter:	Hakan Aksu Institut für Informatik

Koblenz, im Mai 2016

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Contents

Abstract

The publication of open source software aims to support the reuse, the distribution and the general utilization of software. This can only be enabled by the correct usage of open source software licenses. Therefore associations provide a multitude of open source software licenses with different features, of which a developer can choose, to regulate the interaction with his software. Those licenses are the core theme of this thesis.

After an extensive literature research, two general research questions are elaborated in detail. First, a license usage analysis of licenses in the open source sector is applied, to identify current trends and statistics. This includes questions concerning the distribution of licenses, the consistency in their usage, their association over a period of time and their publication.

Afterwards the recommendation of licenses for specific projects is investigated. Therefore, a recommendation logic is presented, which includes several influences on a suitable license choice, to generate an at most applicable recommendation. Besides the exact features of a license of which a user can choose, different methods of ranking the recommendation results are proposed. This is based on the examination of the current situation of open source licensing and license suggestion. Finally, the logic is evaluated on the exemplary use-case of the *101companies* project.

Abstrakt

Durch das Publizieren von Open Source Software soll die Weiterverwendung, Verteilung und allgemeine Nutzung von Software unterstützt werden. Dies ist nur durch den korrekten Gebrauch von Lizenzen für diese Art von Software möglich. Dazu werden eine Vielzahl von verschiedenen Open Soruce Software Lizenzen von Organisationen zur Verfügung gestellt, aus welchen der Entwickler wählen kann, um den Umgang mit seiner Arbeit zu regulieren. Diese werden im Rahmen dieser Arbeit näher untersucht.
Dazu werden, nach einer ausführlichen Literaturstudie, zwei allgemeine Fragestellungen detailliert betrachtet. Zunächst wird eine Nutzungsanalyse von Lizenzen im Open Source Sektor durchgeführt, um allgemeine Trends auszumachen und Statistiken zu erstellen. Diese beinhaltet Fragestellungen wie die Verteilung von verschiedenen Lizenzen, die Konsistenz ihrer Nutzung, die Assoziation von Lizenzen über einen bestimmten Zeitraum, sowie ihre Publikation.
Im Anschluss wird das Empfehlen von Lizenzen für verschiedene Projekte thematisiert. Dazu wird eine Empfehlungslogik vorgestellt, welche unterschiedliche Einflüsse auf die korrekte Lizenzwahl einbezieht, um so möglichst passende Ergebnisse zu liefern. Neben den genauen Eigenschaften einer Lizenz, welche der Nutzer wählen kann, wird so auch eine Möglichkeiten der Einstufung vorgeschlagen. Die Grundlage bildet eine umfangreiche Betrachtung von aktuellen Gegebenheiten und Entwicklungen, sowie die zuvor ermittelten Ergebnisse. Abschließend wird die Logik am Beipsiel des *101companies* Projektes evaluiert.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Ralf Lämmel for the guidance and support in all the time of research and writing of this thesis.

My sincere thanks also go to my family and friends for their continuous support and motivation throughout my study.

Chapter 1

Introduction

The following chapter will give an introduction to this thesis. The research context, a motivation, the research problem, the methodology and the structure of the thesis are given below. All guiding research questions are included as well.

1.1 Research Context

The usage and development of open source software is increasing its popularity [1]. Researchers at research institutions steadily develop new tools and programs and publish them in online open source software repositories. Therefore, a developer can choose between one of many licenses, to determine the further usage of his work. Each of these licenses has its own characteristics and therefore advantages and disadvantages, dependent on its intended purpose. This promotes a complex state of affairs. It is the developer's obligation to properly choose a suitable license for his work, which is not well understood so far [2].

1.2 Motivation

The process of choosing a license for an open source project is a non-trivial task, as there are many influences on a license decision. The diversity and wideness of the open source sector, with large and multifaceted projects being developed, promotes this complexity and unclear extent. To enhance future development and licensing processes, it is important to identify latest trends and behaviors. Therefore, it is helpful, to take a look at a

large number of existing projects, as well as projects which are currently being developed. With the *101companies* project, an exemplary large and complex open source software project is on hand. The *101companies* project[1] is developed by the *Software Languages Team Koblenz*[2] at University Koblenz-Landau. It consists of a wiki system that aggregates knowledge about software languages, technologies and concepts. *101companies* is also a software chrestomathy [3], thus, a collection of small software systems useful for teaching programming and software engineering in general. Each software system is called a *contribution* and implements parts of a common feature model to demonstrate specific languages, technologies and concepts. Consequently, *101companies* is a widely distributed and not centralized system with many different repositories and a large amount of derived and distributed artifacts. Currently there is no definite license convention implemented for *101companies*, as it is not clear, how to properly realize a coherent licensing for the great number of separate files and which license is most suitable. The described problem serves as a basis for this thesis, answering general and more abstract research questions in the field of licensing open source software and finally represents a use-case for which there is to recommend a licensing concept. This leads to the research questions in the following section.

1.3 Research Problem

To gain deeper insights on open source software licensing, general research questions are elaborated and answered within the scope of this thesis. Each question contains several sub-questions, guiding to the final result.
The purpose of this thesis is to firstly depict the state of the art in the field of licensing open source software. Therefore, it is of interest to identify current trends and the general scheme of things. The first research question (RQ) hence addresses statistical issues with most up to date data. Results are to generate in an automated manner and aim to have a high degree of representativity.

- **RQ 1**: What is the state-of-the-art in licensing open source software?
  - **RQ 1.1**: What is the distribution of licenses in the open source sector?
  - **RQ 1.2**: Are there changes in the association of licenses over time?
  - **RQ 1.3**: How consistent is the usage of licenses?

[1] http://101companies.org/ [2] http://softlang.wikidot.com/

  - **RQ 1.4**: How can licenses be publicized properly?

Subsequently, the possibility of recommending licenses for open source software projects is investigated. Through recommendation of licenses, the general usage of open source software, from a developer's perspective, as well as a user's perspective, can be improved. To provide a reasonable possibility to recommend licenses, a logic is to be implemented. Therefore, also the results of the first research question are included. Along the development of the recommendation logic, the following research questions are answered.

- **RQ 2**: Can we provide a recommendation logic for licenses?
  - **RQ 2.1**: How can licenses be classified?
  - **RQ 2.2**: What is the developer's perception on the field of licensing and which inferences regarding license recommendation can be made?
  - **RQ 2.3**: How can licenses be suggested or recommended?
  - **RQ 2.4**: How can a recommendation logic apply to complex open source projects like the *101companies* project?

1.4 Methodology

This thesis is elaborated by processing the above research questions step by step. Generally, the treatment of the research questions is established on an extensive literature research throughout the field of licensing and other correlating fields of research. Afterwards, the research questions are answered systematically, according to modern research principles. A detailed methodology can be found in the beginning of each of the chapters, elaborating the different research questions. Chapter *License Usage Analysis* for the first research question and chapter *License Recommendation* accordingly for the second research question. All implementations are documented with comments in the source files and described in the respective chapters.

1.5 Thesis Structure

The thesis is structured in the following way. The general problem context, the task of the thesis and an overview over the general procedure were given in this introductory chapter. The following chapter *Background* will provide a basic knowledge foundation

in the field of open source software, mining software repositories and the licensing of software, as well as the *101companies* project. The next chapter, chapter three, will deliver a comprehensive overview over literature related to this thesis. The following two chapters provide the answers for the previously presented research questions in depths. The last chapter will finally provide some concluding remarks and an outlook on future work. Additional information will be provided in the appendix, followed by the bibliography.

Chapter 2

Background

The general background knowledge for the following chapters will be provided in this chapter. At first, general concepts and terms will be introduced, followed by an introduction to mining software repositories and a brief overview over terminologies of the *101companies* project.

2.1 Open Source Software

In [4] open source software is described: “Open source software is software whose source code is available for modification or enhancement by anyone”. The term *Open Source Software* originated 1998 along with the foundation of the Open Source Initiative (OSI) and is the dominant term since then [5]. Brügge et al. name four characteristics for open source software:

- License of the software
- Non-commercial attitude
- High degree of collaboration in the development process
- Geographical distance of the developers

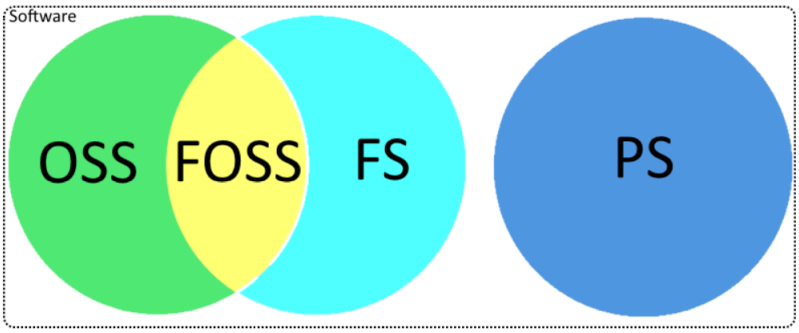
The *Bundesamt für Sicherheit in der Informationstechnik (BSI)* lists four criteria to open source software as well [6]:

- The program can be used without limitations

- It is permitted to analyze and change the program to own needs
- It is permitted to redistribute copies of the program
- The program can be improved and improvements can be distributed

As those definitions have considerable differences, for this thesis open source software is defined as following: *Open source software is unrestricted publicly accessible software or code, published under a copyright and terms of usage and distribution (typically a license).*
This definition concludes, that although the source code of open source software (OSS) is accessible for everyone, the software itself does not have to be (for) free to use it. If so, it is often called free and open source software (FOSS), or free software (FS), where FS is not necessarily open source. For every type of software, there are several different licenses or usage agreements - the focus in this thesis is on open source software licenses (see section 2.2 and appendix), which are eligible for the software owner. Note, that unlicensed software can be open source as well, if the source code is published, as no license is also a regular form of licensing.
The opposite of open source software is proprietary software or closed source software. Proprietary software is always licensed under terms of use, restricting the usage, analysis, distribution and modification. The source code of the software is not publicly accessible and protected by the manufacturers copyright. The term proprietary software does not imply that the software is consequently a commercial software. The relations of the different types of software are illustrated in figure 2.1. For this thesis, open source software and its licenses embody the basis of all research questions (green and yellow in figure 2.1).

Figure 2.1: Software relationships: Open Source Software (OSS), Free and Open Source Software (FOSS), Proprietary Software (PS)



2.2 Software Licenses

Software licenses are legal instruments to ensure the originators copyright and regulate the distribution and usage. A license agreement between the user of software and its originator governs all legal aspects of usage and redistribution, this can include limitations of liability, warranties and disclaimers. In general, software licenses can be fit into two categories: proprietary software licenses and (free- and) open source software licenses.
For proprietary software, the end-user license agreement (EULA) typically grants the use of a specified amount of copies of the software, while the actual ownership remains with the manufacturer. The EULA often contains terms to limit the number of installations allowed and the terms of redistribution.
Licenses for open source software can be divided in permissive and copy-left licenses. Permissive licenses aim to have minimal requirements on the redistribution of the software, whereas copy-left licenses ensure that all subsequent end-users receive the specified rights.
The GNU General Public License[1] (GPL), provided by the *Free Software Foundation*, gives an example for a copy-left software license. GPL intends to guarantee the users freedom to share and change a program and all of its versions, by assuring it to by free, in terms of freedom, for all users. Modification, analysis and redistribution of software or software parts is possible, if the user adheres to the terms and conditions of GPL.
An example of a permissive software license is the *Berkeley Software Distribution (BSD)* license. BSD gives the user the permission to analyze and privately modify the software, as well as redistribute it with minimal requirements. By using the BSD license, a user has the permission to use code as a part of proprietary software.
Another classification for open source licenses is given by [7]. He divides licenses in four categories, which specify the previous classification more precisely:

1. permissive: permit the software to become proprietary
2. weakly protective: weak copy-left: prevention the from becoming proprietary, yet permitting software to be part of a larger proprietary programs
3. strongly protective: strong copy-left: prevention from becoming proprietary
4. network protective: protection of networks included

[1] http://www.gnu.org/licenses/gpl-3.0.de.html

Alternatively, [8] classifies licenses by self-explaining slogans, which are based on the above classifications in a more comprehensible way. *I want it simple and permissive* for permissive licenses, *I am concerned about patents* for permissive licenses with patent regulation included (e.g. Apache License) and *I care about sharing improvements* for weakly to strongly protective licenses. However, all those classification approaches of open source software licenses tend in the same direction. They are the most common ways of classifying licenses, also in regard of RQ 2.1.

Generally, open source software licenses can have different features [8]: requirements, permissions and constraints. Those features can be defined the following way:

- Requirements: What the license requires users and license holders to do.
- Permissions: What license holders allow users to do by choosing the license.
- Constraints: What license holders restrict users to do by choosing the license.

The tables 2.1, 2.2 and 2.3 give an overview over those features, which are probably not exhaustive. Generally, software licenses for open source software are the basic point of investigation for this thesis, in all possible categories.

Table 2.1: Requirement features of licenses

Requirement	**Explanation**
License and copyright notice	The code has to provide a copyright and license notice
State changes	Significant changes within the code have to be annotated
Disclose source	Source code must be made available when distributing the software
Network use	Users who interact with the software via network have the right to receive a copy of the corresponding source code
Library usage	The library may be used in non open source projects
Same License	Modifications must be released under the same license (sometimes related licenses)

Table 2.2: Permission features of licenses

Permission	**Explanation**
Commercial use	The software may be used for commercial purposes
Private use	The software can be used and modified without distribution
Distribution	The software may be distributed
Modification	The software may be modified
Patent use	The license provides an express grant of patent rights from distributor to the recipient
Sub-licensing	Sub-licenses can be provided to third parties not included in the license

Table 2.3: Constraint features of licenses

Constraint	**Explanation**
Hold liable	No warranty and liability for the owner
Use trademark	Does not grant right in the trademark
Sub-licensing	No sub-licenses are allowed
Distribution	Distribution of the software is not allowed
Modification	Modification of the software is not allowed
Patent use	No rights in the patents of contributors (explicitly stated)

2.3 Mining Software Repositories

The general idea of mining software repositories is described in [9], the mining software repositories conference of 2015, as following. “The mining software repositories (MSR) field analyzes the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. Software repositories such as source control systems, archived communications between project personnel, and defect tracking systems are used to help manage the progress of software projects”. They see the main goals of MSR in supporting the maintenance of software systems, improving design and reuse of software, empirically validating novel ideas and techniques, understanding software development and evolution and planning future development. In this thesis those goals will be investigated in regard of the open source licensing field. Several mining software repositories methodologies are applied in the license usage analysis.

2.4 101companies

The *101companies* project was briefly described in the introductory chapter as a comprehensive knowledge and data collection. It is organized as a widely distributed and not centralized system with many different repositories and a large amount of derived and distributed artifacts. [10] gives a more detailed description. Relating to this thesis, different properties of the project have to be lined out, especially the great number of software technologies, software languages and stakeholders. As an academic project, it is under steady development and aims to grow in all areas.
The following terms are the foundation to understand the structure of the *101companies* project. To deal with versatile influences on the project, the notion of *contributions* is introduced, meaning the implementation of a software system, to better apply to varying stakeholders and objectives. Different contributions are grouped in *themes*, which are tailored towards interests of specific stakeholders to help users to navigate through the knowledge collection. The *101companies Ontology* classifies all entities, that are relevant for the *101companies* project and is divided into different categories, which are broken down into three sub-categories: programming technologies, software languages and technological spaces. Those categories help to cluster different parts of the system and can therefore provide knowledge about those parts.
The *101companies* project is introduced, as it will be referred to as a general research motivation and a complex exemplary use-case and example, to apply a coherent and reasonable licensing to, effectuating the research results of this thesis. The contribution-structure, in which the contributions are distributed throughout repositories, will be a challenge for license usage analysis and a coherent licensing concept.

Chapter 3

Related Work

In the chapter *Background*, general concepts and terminologies were introduced. The following chapter will give a more detailed overview over different aspects of licensing open source software, as a basis for the following elaborations. Also, the necessary background information for the experimental and analytic methodologies are provided.

3.1 Current Open Source Licensing

The licensing of open source software is a field on which every developer, who aims to make his work publicly accessible, has to inform himself. Especially the process of choosing an appropriate license is indispensable, if others shall profit from it. The following sections give an overview over literature regarding this process and other general aspects in the context of this thesis.

3.1.1 Choosing an Open Source License

The right choice of a license for an open source project is the key to ensure the intended usage of the software. The first step in choosing a license is to think of the general way the owner wants to distribute his software. Therefore, it is helpful to look at the classification approaches introduced in the background chapter from the perspective of making a choice.
In short, [11] differentiates between three types of licenses: *Free-for-all licenses*, *Keep-open licenses* and *Share-alike licenses*. Whereas the *Free-for-all licenses* only require the user to give credit to the original author, the *Keep-open licenses* also require, that modifi-

cations have to be published as open source as well. The *Share-alike licenses* refer to the copy-left licenses. When software is modified or extended, the result as a whole has to be made available as open source as well. Since all of those three types have its benefits and drawbacks, [11] describes the problem as a choosing between acceptance and contribution.

There are several ideas of how owners can be helped to decide for the right license. *Choosealicense.com*[1] is probably one of the most common license suggestion websites, as it is linked on *GitHub*. They refer to the three types of licenses with short slogans to describe them roughly. If an author wants to publish software "simple and permissive" they suggest the MIT license. If the author is "concerned about patents" the Apache License and if the author "cares about sharing improvements" the GPLv2 or GPLv3 license. If more choices are needed, *choosealicense.com* provides a lookup table of the most popular licenses including their requirements, permissions and constraints. This way, users can make a quick choice, but can also inform themselves more deeply.

Within the scope of this thesis, it is a goal, to provide a recommendation logic for licenses (RQ2). It is to see, if the named approaches are a good way to help authors choose a license, or if it is possible to develop a different kind of recommendation logic.

In [12] the developer membership, coding activity and development speed in relation to the license choice of a project was investigated. They were able to demonstrate that the social movement account of voluntary participation in open source projects does seem to predict the relationships between the development process activity and the license choice. Specifically projects with a copy-left license are associated with higher developer membership and coding activity and suggest evolving faster. This indicates that license suggestions can go further than just involving the intended usage of the open source software: additional factors within the process of choosing a license might be relevant. Possibly, including additional factors into the suggestion process might even result in a recommendation logic, which would be more useful than just a suggestion, for example legal factors (see following sections).

3.1.2 Open Source License Distribution

The license distribution on *GitHub* was analyzed in year 2015 by *GitHub* itself [13]. Table 3.1 shows the analysis results.

Those results can be used as a comparative value, as the goal of this thesis also aims

[1] http://choosealicense.com/

Table 3.1: License distribution on *GitHub*

License	Distribution [%]
MIT	44.69
GPL	24.19
Other	15.68
Apache	11.19
BSD	6.23
Unlicensed	1.87

to analyze *GitHub* projects. It is important to note that their approach only considered a project as licensed, if the license was declared in the licenses API of *GitHub*. If there are license files or the license is declared in each file of the project, but not in the API, this approach is not sufficient. This problem is considered within the scope of this thesis.
The private company *Black Duck* made a list of the most used licenses in the open source sector as well, using their knowledge base. However, the overall analysis process is not further mentioned [14]. The results of their analysis are illustrated in table 3.2.

Table 3.2: License distribution according to *Black Duck*

License	Distribution [%]
GPL	38
MIT	24
Other	16
Apache	16
BSD	6

3.1.3 Usage of Licenses Over Time

As mentioned in the section *Open Source License Distribution* before, *GitHub* investigated usage of licenses on their platform *github.com*. They were also able to make some statements on the usage of licenses over time [13]. The results are shown in figures 3.1 and 3.2.
Figure 3.1 shows a graph of licensed repositories over time with a decreasing percentage of licensed repositories. The reason for this trend is not further mentioned, however the sharp spike in 2013 shows, that users actually use license recommendation tools, as

GitHub's choosealicense.com was started then. This is a general indicator, that recommendation tools are useful and justifies further research. Figure 3.2 approves this statement, as it shows that the three featured licenses record a swift uptick in 2013, with the relative percentages remaining otherwise steady over the past six years.

Figure 3.1: Percentage of licensed repositories on *GitHub* [13]

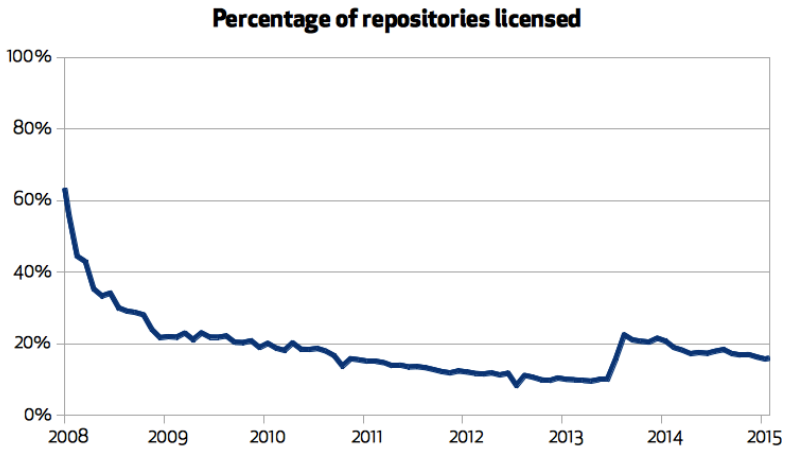
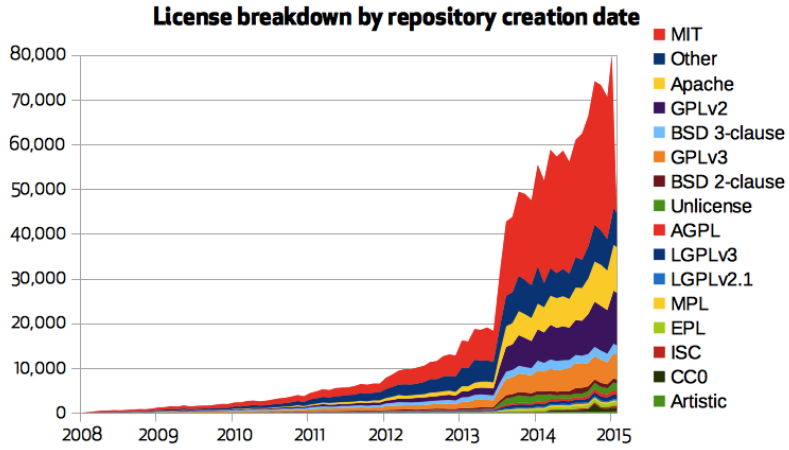


Figure 3.2: License breakdown on *GitHub* [13]



3.1.4 License Enforcement and License Publication

A software license is a legal measure to ensure the software owners rights and declare how his work is allowed to be used. It is essential to take a brief look at the legal aspects

of licensing, as it corresponds with other licensing concepts and is the basis of a solid licensing concept. When it comes to enforcing an open source license, [15] state several issues. It is not clear, whether a license is to be treated as a mere copyright or as a contract between owner and user and it is unclear to what degree a court will actually enforce an open source license, as it is not a common case yet.
The *Free Software Foundation* (FSF) in collaboration with the *Software Freedom Conservancy* (SFC) published the *The Principles of Community-Oriented GPL Enforcement* [16]. They state, that license enforcement should only serve the society and not financial motives. FSF and SCF represent authors and copyright holders and aim to support their interests. To enforce copy-left licenses, they want to help users to comply with a correct license usage. Lawsuits are only a "last resort". Karen Sandler, Executive Director of the *Software Freedom Conservancy* reasons their approach by saying: "The ugly truth about copyleft compliance is that if there are never any lawsuits when companies refuse to comply, then there's very little incentive to do the right thing" [17].
Those sources illustrate that there is a need for clarification in the field of license enforcement, as authors and copyright holders are mostly self-responsible in enforcing their rights. However, there have been verdicts [18] and law offices are willing to support their clients. A problem is, that it is hard to discover license violations and to distinguish between code reuse or self-developed code, at least for smaller code parts [19].
To enable the enforcement of a license, the license has to be included correctly. How to include a license is often explained in the full license text, or on the originators' website. *GNU* for example recommends attaching a short notice to the program, ideally to the start of each source file, or at least the "copyright" and a pointer to the full notice in every source file [20]. The short notice for the GPL licenses looks as follows:

'one line to give the program's name and a brief idea of what it does.' Copyright (C) 'year' 'name of author'

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

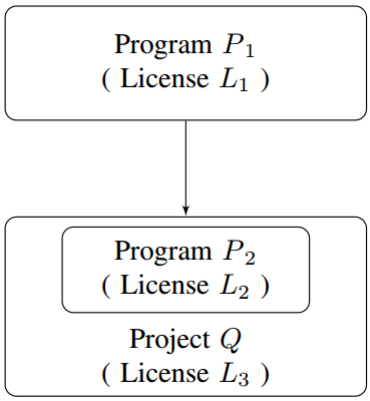
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see 'http://www.gnu.org/licenses/'.
Other types and forms of publicizing licenses are investigated in the chapter *License Usage Analysis*. The enforcement of licenses is also included in the development of a recommendation logic for licensing open source projects (see chapter *License Recommendation*).

3.1.5 License Compatibility and License Combination

Besides the publication or declaration of a license, another aspect for correct and enforcible licensing is essential: the compatibility and combinability of licenses. In the field of open source software licenses, few licenses are dominantly used. Those licenses tend to be compatible, so that software can be combined for larger projects, although those projects already have one or more different licenses. However, it is possible, that issues arise, when licenses are applied to already licensed software or software packages, as licenses can contain contradictory features. [21] prove that license violations due to incompatibility are not uncommon. They define a license incompatibility or violation as illustrated in figure 3.3. Therefore, let program *P1*, licensed under *L1*, be reused in the form of program *P2* licensed under *L2*, which includes *P1* and all derivative works, if any. *Q*, the project which contains *P2* may by itself, have an overall license *L3*. Based on these notations, they define violations by the following rules:
1) Type 1: *L1* and *L2* are incompatible or, *L1* and *L2* are compatible, but *L1*s copy-left nature is not honored by *L2*.
2) Type 2: Similar to the violations of first type, but with checks between *L2* and *L3* instead.
[7] investigated the compatibility of the most common licenses with an emphasis on the combinability and created an illustration (fig. 3.4). Wheeler describes, that the different licenses are represented by the named boxes. An arrow from box A to box B means that you can combine software with these licenses; the combined result effectively has the license of B, possibly with additions from A. To see if software can be combined, just start at their respective licenses, and find a common box you can reach following the arrows. Therefore, following the paths shows the compatibility of licenses. Some minor popular licenses (as MPL1.1) show a limited compatibility to other licenses, so it is important to note that not all licenses are necessarily compatible to each other and can even contradict.

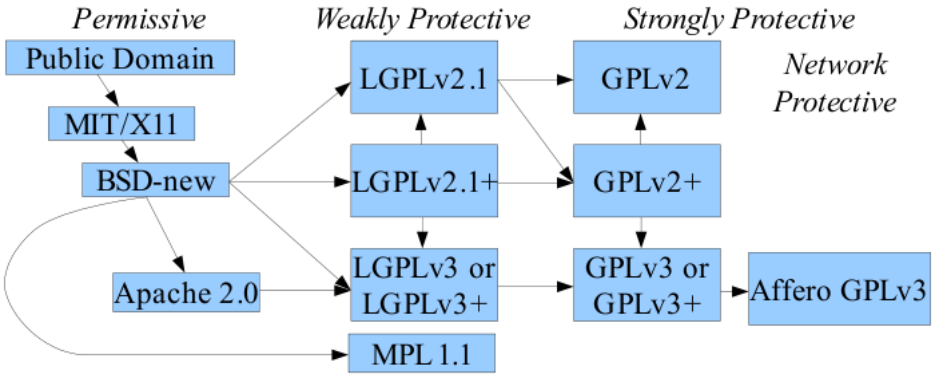
Figure 3.3: Definition of license violation [21]



This shows that the compatibility of licenses is not sufficient to identify license violations, the combinability has to be considered as well, which is not often accomplished.
In short, for this thesis, compatibility and combinability are defined the following way.

- If two or more licenses do not contradict each other, they are compatible.
- If wo or more licenses can be combined under one license, they are combinable (includes correct declaration).

Figure 3.4: Compatibility and combinability of licenses [7]



3.2 Mining Licenses for Analytical Purposes

To analyze the state of the art in current open source licensing, it is necessary to take a look at current licensing standards. To examine existing open source projects regarding their licensing, a methodology was developed, to automatically download software from online repositories and analyze its license information.
The methodology was determined within the scope of a seminar, held preceded to the composition of this thesis. The goal of the seminar was a systematic literature research in the past MSR conferences[2], to identify possible research questions and an experimental methodology in the field of mining software repositories. The literature research identified the following related work, leading to a final methodology presented in the next chapter.

3.2.1 License Identification

In *A Method to Detect License Inconsistencies in Large-Scale Open Source Projects* [1] open source software was examined regarding license inconsistencies. Referring to the experimental implementation of this thesis in the context of RQ1, some parallels can be seen, as they needed to analyze license information as well. They used the license classifier *Ninka*[3] to analyze source code automatically and in an efficient way [22].
The developers of *Ninka* implemented the following procedure model:

1. The *CommentExtractors* extract the top comments of each source file. If no comment extractor is known for the files programming language, it extracts top lines from the source (currently the first 700 lines)

2. Ninka works by matching sentences of licenses, hence it needs to properly break text into sentences.

3. The *SentenceFilter* checks if sentences are related to a license or not. It is valuable to know if a file contains lines that look like a license or not (e.g. to know that a file has no license).

4. Then it creates a file that corresponds to the recognized sentence tokens. For each sentence, it outputs its sentence token, or unknown otherwise.

5. Finally it looks at the sentence tokens and outputs the licenses found.

[2] msrconf.org

[3] http://ninka.turingmachine.org/

An alternate approach was presented in [23]. They proposed a methodology to analyze Java *Jar* archives regarding their licenses with a code-search approach. They implemented the following steps:

1. Extract data of the jar archives by decompiling
2. Identification and classification of licenses saved in text files
3. Querying of license information decompiled from the byte code via code search engines, such as *Google Code Search*[4] to identify associated licenses
4. Comparison of the results

Ninka was used by [23] as well, to verify their results.

3.2.2 Automated Data Assembly

For the automated approach it is necessary to implement a system to automatically download free open source projects and provide them locally. Especially with regard to the representativity of the research results, it is essential to retrieve a large amount of license information.
In [24] different repositories containing open source software collections are presented. Besides *Debian*[5],*Maven*[6] and *Drupal*[7], *GitHub*[8] is one of the largest software collections. *GitHub* will be in focus of this experiment. Other repositories that should be named in this context are *bitbucket*[9], *sourceforge*[10] and *Google code*[11]. Possibly some inferences to the domain of application or other assumptions on the software can be made by their source repository or licensing information as well.
[25] addresses the collection process of data from *GitHub*. *GitHub* limits the amount of requests from a host via the *GitHub API* within a 60 minutes interval. *GitHub Archive*[12] could be an alternate way of investigation, as *GitHub Archive* spares said limitation. *GitHub Archive* describes itself as a project to record the public *GitHub* time line, archive it, and make it easily accessible for further analysis.
A possible approach to extract data from *GitHub* is presented in [2]. To collect the needed

[4]https://code.google.com/
[5]http://www.debian.org/
[6]https://maven.apache.org/
[7]https://www.drupal.org/
[8]https://github.com/
[9]https://bitbucket.org/
[10]http://sourceforge.net/
[11]https://code.google.com/
[12]https://www.githubarchive.org/

data from *GitHub* they made a script to generate a comprehensive project list of projects in *GitHub*, using the *GitHub API*. As the first list contained over 12 million projects, the list was filtered. All desired projects were then downloaded via the *git-clone* command. Their local storage contained projects of more than 6TB. Afterwards they analyzed the cloned data with an analysis tool in relation to the *MARKOS European Project* [26]. The tool used the license classifier *Ninka* as well.

On the basis of the researched literature, the seminar effort developed the described methodology, which can be summarized as follows:

1. Collection of source data (projects) in open source repositories
2. Extraction of license data from the source data
3. Analysis of the retrieved information

A detailed description of the final methodology is given in the respective chapter in section 4.1.1, *Automated License Mining*.

Chapter 4

License Usage Analysis

As explained before, one goal of this thesis is to depict the state of the art in the field of licensing open source software. Therefore, it is of interest to identify current trends and the general scheme of things. The first research question hence addresses statistical issues with most up to date data. Results are to generate in an automated manner and aim to have a high degree of representativity. Additionally, methodologies and techniques will be helpful for several parts of the next research question. The following chapter will comprise the methodologies, design, implementation and final results of RQ 1. Therefore the following sections elaborationg the subqeustions of RQ1 are ordered, so that they build up on the previous results of each sub-question, as some of the statistics are strongly related.

4.1 Distribution of Licenses

The first part of the first research questions (RQ 1.1) is to elaborate the distribution of licenses in the open source sector. This question aims to identify the most prominent licenses as a basis for further research. Therefore, an approach for automated license mining and identification is presented and performed on a multitude of *GitHub* repositories.

4.1.1 Automated License Mining

The conception process of the general methodology for this part of the first research question, concerning the distribution of licenses in the open source sector, was briefly de-

scribed in the last chapter *Related Work*. The exact approach will be stated more precisely in this section. The analysis of a large amount of open source projects was performed in six sub-steps.

1. Create a search list:
   To realize the first two sub-steps of this approach, a Java program was implemented. For creating a list of open source projects from the open source software archive *GitHub*, the *GitHub API v3* is used as an interface between the program itself and *GitHub*. The program requests search queries for repositories with specific attributes. As for some parts a random distribution of projects was sufficient, a list of projects containing the letters "A" to "Z" was created automatically, it is however possible, to further specify the search queries. *GitHub* limits the search requests to 1.000 search results, so, after removing all duplicates, the list can contain about 26.000 open source projects at max. Different search preferences can hence extend this list.

2. Save open source projects locally:
   After creating the search list, the program automatically starts to *clone* the projects from the repositories, which means to locally store the projects of the list, as fast as *GitHub* and the available bandwidth allows it to. To preserve the representativity of the data, empty projects and *GitHub* related data (e.g. the .git folder) are deleted. The manual deletion of particularly large files that do not contain any license information improves the performance of the following steps additionally.

3. Analyze open source projects:
   As mentioned in the chapter *Related Work*, the *Ninka* classifier was used to analyze the project data for licenses. The console application was run on *Ubuntu 14.04* with the following command:

   console command (shortened):

```
1  find * | xargs -n1 -I@ ./ninka.pl '@'
```

4. Preprocess the data of the analysis:
   Ninka creates a set of analysis files for every source file existing in a project. Those files have to be preprocessed, as the amount of files for even one project easily

rises above thousands of single files. Therefore, a ruby script was implemented. The script searches for all ".license"-files created by *Ninka* and investigates them for the licenses that *Ninka* detected. The classified licenses can be found in the appendix. For every different license that is found in the project, a "license.txt"-file is created at the root directory of the project. Note that it is important that every license is only counted once within a project, to obtain correct results in the evaluation. This is not self-evident, as a license declaration is often included in every source file explicitly. Therefore, the ".txt"-files are overwritten by identical results, to ensure the correct count. The intermediate creation of ".txt"-files proved itself as a helpful strategy to split computing effort and facilitate reproducibility. Additionally it creates a clear overview.

5. Evaluate the preprocessed data:
   To finally evaluate the results of the previous steps, the occurrences of each license are counted. The addition of those values leads to the distribution of licenses within the examined projects and can then be summarized for all projects. The implemented ruby scripts can be seen in the appendix.

6. Interpretation of the results:
   The final step is then to illustrate and interpret the results. This will be done in the sections 4.1.3 and 4.1.4.

4.1.2 Design and Implementation

As previously described, the automated approach of identifying licenses required the implementation of a Java program. This program and the additional scripts are presented in this section.

GitDownloader

The program *GitDownloader* was implemented, to automatically clone open source projects from *GitHub*. It consists of three packages. Package *main* contains the main method and all necessary define parameters, to adjust the final output and retrieval process. The *searchCrawler* builds search queries for the *GitHub API* which are then used by the *gitCloner*, cloning the queried projects from *GitHub* on a local disc space.
The main method gives a good example of how the program is executed, the preferences can be seen the *Defines* class, also containing detailed comments on the preferences (see

appendix).

Main.main.java (shortened):

```
1  public static void main(String[] args) {
2          crawler = new Crawler();
3          crawlerDataFilter = new CrawlerDataFilter();
4          repositoryCloner = new RepositoryCloner();
5
6          File searchResults = new File(Defines.searchResultFile);
7          boolean created = searchResults.createNewFile();
8
9          crawler.grabContent();
10         crawlerDataFilter.removeDuplicates();
11         repositoryCloner.cloneWithTimeout();
12 }
```

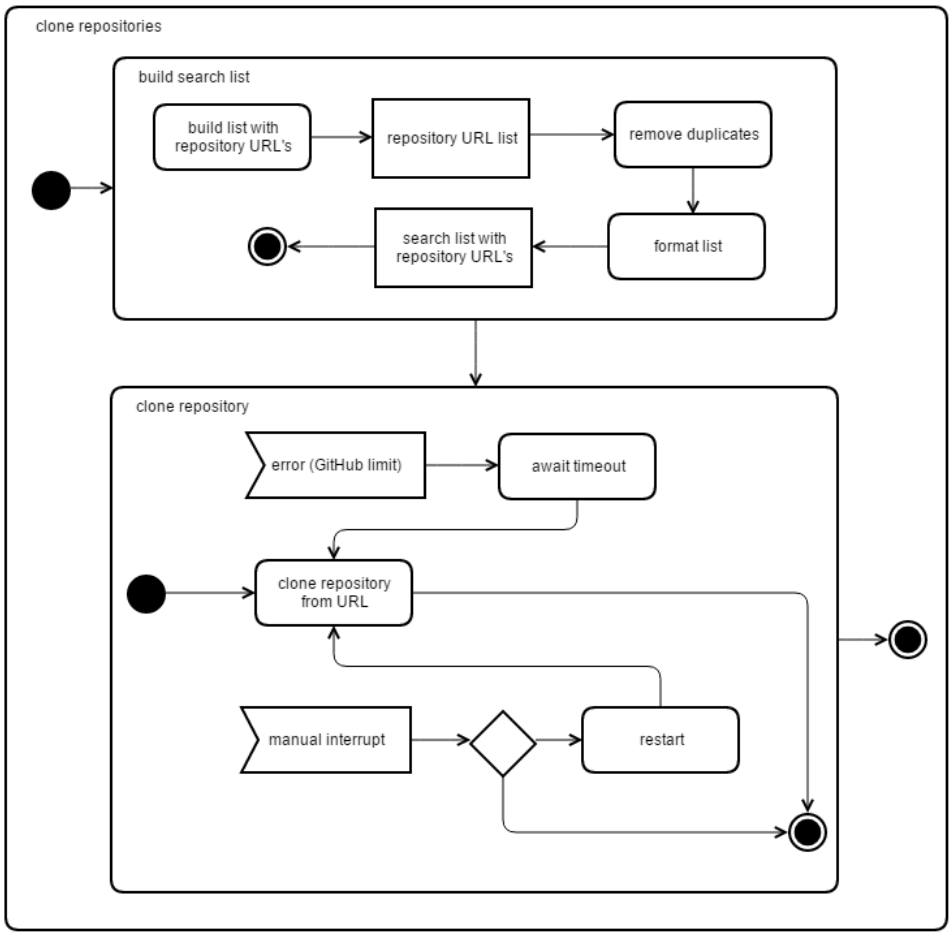
In lines 2 to 8 of the main function, all necessary objects and files for the later usage are created. After the execution, the file *searchResults.txt* lists all projects retrieved by the *crawler*, which grabs the results of the search in the *GitHub API*. As duplicates in the search cannot be precluded, the list has to be filtered again by the *crawlerDataFilter*, which also adjusts the formatting of the list. Finally, the list is used to clone the project repositories directly from *GitHub* as shown below.

gitCloner.RepositoryCloner.java (shortened):

```
1  public Object cloneRepository(){
2          File localPath = null;
3          Git result = null;
4          repoUrlBuilder = new RepoUrlBuilder();
5          while(repoUrlBuilder.loadNextRepo()){
6                  localPath = File.createTempFile("TestRepository", "");
7                  localPath.delete();
8                  try {
9                          result = Git.cloneRepository().setURI(repoUrlBuilder.
                               getCurrentRepo()).setDirectory(localPath).call();
10                 } catch (GitAPIException e) {
11                         e.printStackTrace();
12                         cloneWithTimeout();
13                 }
14                 result.getRepository().close();
15         }
16         return result;
17 }
```

At first a new folder is prepared for the cloned repository to store (here: temporary folder). Then the clone process begins by downloading the repository from the URL that was formerly retrieved. If an error occurs, the cloning process is restarted after a timeout. Errors occur, as *GitHub* limits the amount of requests. Working with timeouts generally proved itself as an expedient strategy to work around the limitations. All timeouts can be adjusted manually in *Main.defines.java* (see Appendix for all defines). The program finishes when all repositories in the list are cloned or it is interrupted manually in before. The UML activity diagram in figure 4.1 illustrates the overall process. The full code of the *RepositoryCloner* will be attached to this thesis.

Figure 4.1: UML activity diagram for the GitDownloader



Additional Scripts

To analyze the output of the *Ninka* license classifier, the *ruby* script *process.rb* was implemented. To understand the idea of the script, it is important to know, that *Ninka* creates analysis files for every file successfully analyzed, for example "filename.license"-files. Those files contain keywords for every license *Ninka* was able to classify. To process all those files, the script *process.rb* runs through all directories and reads the keywords from every "filename.license"-file. This is to be done for every project that was analyzed by *Ninka* in before. Only if a license is found in a project, the script creates another countable file in the root folder of each project once for every license. A second script *count.rb* can then count the total occurrences of licenses in a set of projects. This way it can be excluded, that licenses are counted more than once within a project. For example: assuming that a project declares a license more than once, e.g. in every source file header, the license is counted only once for the project. A third script (*combinations.rb*) lists all license combinations for each repository. The scripts *count.rb* and *process.rb* can be seen in the appendix as well.

4.1.3 Results

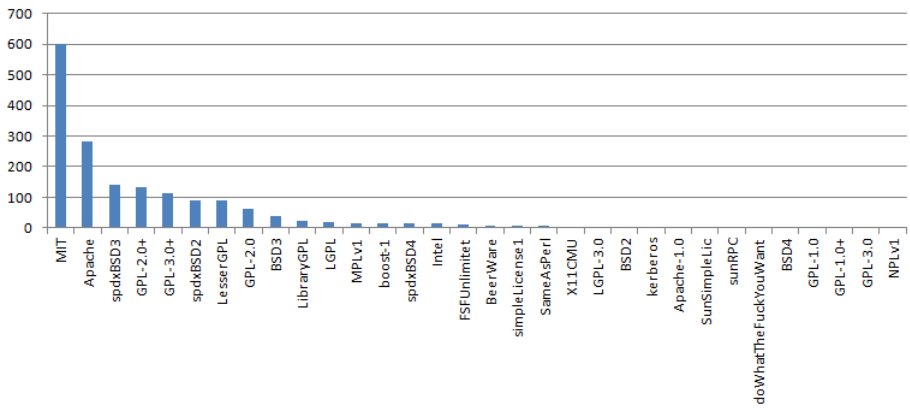
Respective to the described methodology and implementation, the experimental approach lead to the following results for the first research question (RQ 1).
The first run of the experiment contained 1,330 random open source projects from *GitHub*. Those projects contained 1,722 licenses, without doubling within a project. The number of retrieved licenses is higher than the number of overall projects, as a project can have multiple licenses. For one project, each license is counted only once. The first run of the experiment aimed to create a reference result for the following runs and provide a basic knowledge base. The results of the first run can be seen in table 2 in the appendix, and in a combined version in table 4.1 below. The combined results combine the results of different licenses with the same provenance (e.g. BSD combines spdxBSD(1-4) and BSD(1-4)). The selected licenses comprise clearly more than 90% of all licenses used for open source projects.
For more information on specific license identifiers, used in the tables, also see [27].
The full results of the first run are illustrated in figure 4.2, whereas figure 4.3 illustrates the simplified combined results. Those results are created from the total number of identified licenses. The results from a project's perspective, including the unlicensed projects, can be seen in figure 4.4. Therefore, the distribution of each license is broken down to its

Table 4.1: Combined license distribution of all classified licenses on *GitHub*: (Run1)

License	Distribution [abs]	Distribution [%]
MIT	599	34.79
GPL	454	26.36
BSD	286	16.61
Apache	285	16.55
Others	98	5.69

proportion including the unlicensed projects. Generally, only licenses that were classified at least once, are listed in the overall statistics.

Figure 4.2: Graph of the license distribution of all classified licenses on *GitHub*: (Run 1)



The second run of the experiment contained 842 open source projects, likewise from *GitHub*. The approach was generally the same, but the set of projects was specified differently. As for random projects, the reproducibility is difficult and the representativity is probably not optimal, the second run used a set of projects which were highly rated (most "stars" in ascending order from "A" to "Z") from the *GitHub* community. Those projects contained 1,206 licenses without doubling within a project. Again, the number of retrieved licenses is higher than the number of overall projects, as a project can have multiple (different) licenses. For one project, each license is counted only once. The second run had the combined license distribution as can be seen in table 4.2, the full results can be seen in table 3 the appendix. Table 4.3 shows the projects perspective.
Although the experimental results should be representative, some general limitations of

Figure 4.3: Diagram of the combined license distribution of all classified licenses on *GitHub*: (Run 1)

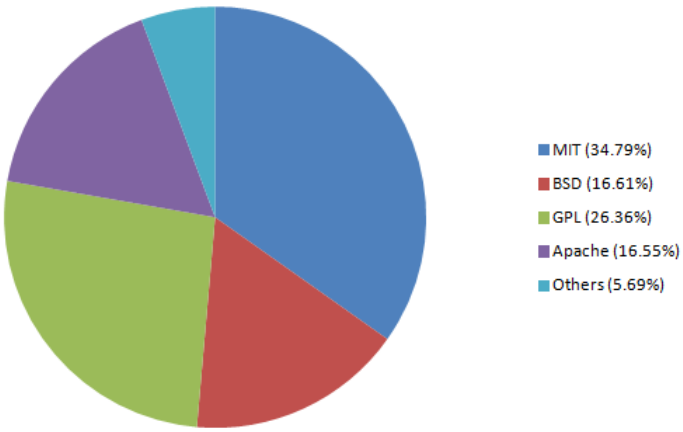


Table 4.2: Combined license distribution of all classified licenses on *GitHub*: (Run2)

License	**Distribution [abs]**	**Distribution [%]**
MIT	583	48.34
Apache	221	18.33
BSD	196	16.25
GPL	167	13.85
Others	39	3.23

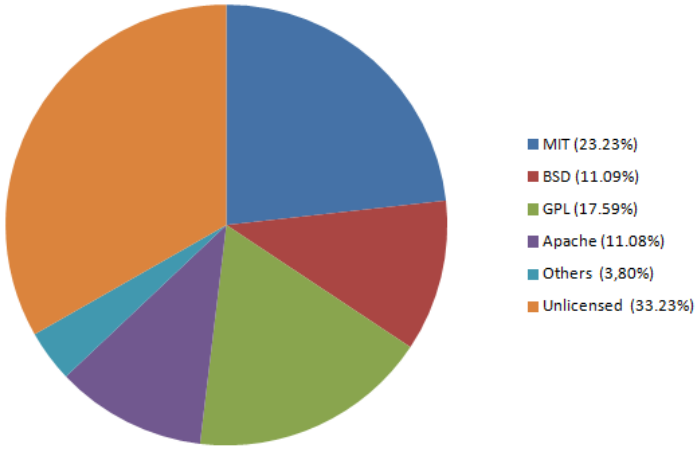
Table 4.3: Combined license distribution of projects on *GitHub*: (Run2)

License	**Distribution [%]**
MIT	41.57
Apache 221	15.77
Unlicensed	13.99
BSD	13.98
GPL	11.91
Others	2.78

the described approach must be noted.

1. *Ninka*: Referring to [22], *Ninka* has an accuracy of 93%. Although this is probably the highest accuracy for current license classifiers, it leaves space for slight inaccuracies in the results. It is also possible, that some less popular licenses are not classified by *Ninka*. All classified licenses can be seen in the appendix.

Figure 4.4: Combined license distribution of projects on *GitHub*: (Run 1)



2. Projects without licenses: With said approach it is not possible to make a clear statement of how many projects do not have any license. This has three reasons.
   First, if a project does not have any license, but a library the project uses has, the license is counted for the project, as it is not possible to differentiate between a project itself and content of foreign sources.
   Second, projects on *GitHub* are not necessarily finished and in some state of working copy. Licenses might be added or deleted any time. Some projects only contain README files, that generally do not contain any license header and hence keywords. And some might only have noted the used license in the *GitHub API*.
   And third, if *Ninka* does not classify a licensed project, because the license is not used in the classifier.
   However, it is possible to identify the projects where *Ninka* was not able to classify any license. This was the case for 442 out of the 1,330 projects, making it 33.23% of all projects of the first run and respectively 13.99% in the second run, not containing a license, as a low estimate.

4.1.4 Evaluation

The results of this thesis approach to identify the usage of licenses on *GitHub*, were presented previously. To evaluate those results, it is sensible to compare them to other research results (also see Table 4.4). In the chapter *Related Work* an alternate approach of mining license usage via the *GitHub API* was presented.

For the first run, both approaches have in common, that the MIT license is the most used license on *GitHub*, although with noticeable lower percentage in the thesis results, followed by the different GPL licenses and with some space in between the Apache licenses. Another difference is, that the BSD licenses are more popular in the thesis results, than in the related approach, comparable with the Apache licenses usage. The higher popularity of BSD licenses can however be approved by other sources [28] which use *GitHub* as a knowledge base. *Other* licenses, by contrast, are less popular in the thesis results. Moreover, the API-based approach of [13] has a much lower percentage of unlicensed projects, as the API usage logically connotes the selection of a license. Since [13] do not clarify which licenses they included, it is impossible to make further comparisons for this section. As a side note, a minor influence for the difference of the unlicensed project count might be caused by *Ninka*, as the less popular licenses might not have been classified, this effect should however be valuated low, as more than 75 licenses were included in the classification process (see appendix for all classified licenses). In general, the results of [13] can nevertheless be approved.
The second related comparative work was from *Black Duck*. Their results differ in the top licenses category, as the GPL and MIT licenses are swapped in their ranking. However, [29] already prognosticated the decrease of GPL usage and an increase of more permissive licenses in the late year 2014. The percentage for the *others* licenses category is lower as well. In contrast, the Apache licenses are really close to [13]. It is important to note, that *Black Duck*'s knowledge base does not necessarily consist of projects from the *GitHub* community, which could be a general explanation of differences and why the first (and second) runs results were closer to *GitHub*'s results. This further approves, that fluctuation in the results is common. Generally, it can be said, that the results of this thesis first run tend in the same direction as the results from the other researches. Approximately all three researches show comparable results, which shows a general confirmation of all approaches.

The second run, including only highly rated projects, had variating results. It is noticeable, that GPL licenses have a significantly lower distribution in those projects than on the other data sets, whereas the MIT license is by far dominating, close to the results of [13]. The usage of Apache licenses and BSD licenses is nearly identical with the thesis first run. The percentage of licensed projects is also markedly higher for the highly rated projects, which seems comprehensible, regarding the community standing and development con-

text. Overall, the results lead to the conclusion, that the second run can be compared to the others as well. Figure 4.5 and table 4.4 show the combined distributions, of both runs, in comparison to the related approaches, which illustrates the previous statement. However, it can be seen that different setups create different results.

All in all, the license distribution analysis reveals, that there is a widely spread distribution of licenses, with few established licenses notably dominating. Although less popular licenses seem to have a certain support, the "big" licenses will continue to be dominating in the near future, probably with the (by tendency) permissive licenses, especially the MIT license, on top, as the second run indicates. Generally, those assumptions are based on the results of two different test runs on *GitHub* and can be approved by related researches.

Figure 4.5: Comparison of the combined license distribution

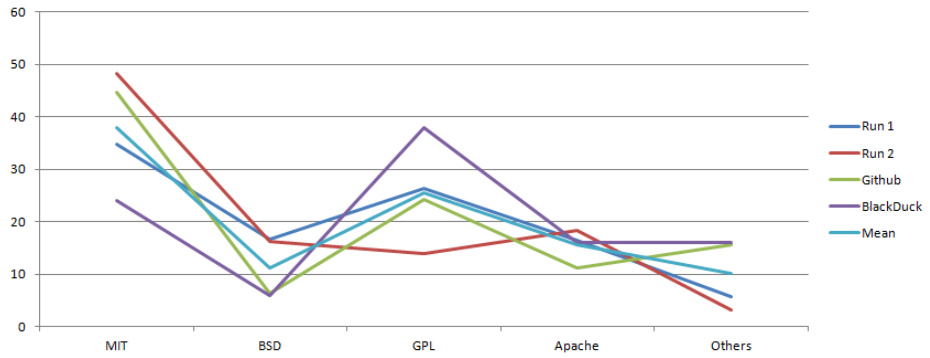


Table 4.4: Comparison of the combined license distribution

License	**Distr. [%]** Thesis Run1	**Distr. [%]** Thesis Run2	**Distr. [%]** GitHub	**Distr. [%]** BlackDuck	**Mean**
MIT	34,79	48.34	44.69	24.00	37.96
GPL	26.36	13.85	24.19	38.00	25.60
BSD	16.61	16.25	6.26	6.00	11.28
Apache	16.55	18.33	11.19	16.00	15.52
Others	5.69	3.23	15.68	16.00	10.15

4.2 Association of Licenses Over Time

The general distribution of licenses over a period of time was already investigated by *GitHub*, as described in the chapter *Related Work*. Another correlating aspect is the association of licenses over a period of time (RQ 1.2), as changes in the licensing can always have different consequences for users and developers.
To get a deeper insight on license changes, the commit messages on *GitHub* were investigated on a large-scaled basis by utilizing the knowledge base of *GitHubArchive.com*. For the year 2014 *GitHubArchive* archived 66,428,295 *PushEvent*'s from the *GitHub* timeline. With the help of *Google BigQuery*, a data warehouse to enable the analysis of massive data sets, those events were investigated regarding their *payload_commit_msg* with the following query:

BigQuery query:

```
1 SELECT repository_name, payload_commit_msg
2 FROM [githubarchive:year.2014]
3 WHERE type="PushEvent"
4         AND (
5                 LOWER(payload_commit_msg) CONTAINS "license" AND
6                 LOWER(payload_commit_msg) CONTAINS "change"
7         )
```

The resulting table contained 11,382 commit messages containing the keywords "license" and "change" (0.017% of all commits). This means, that the changing of a license for a project is not uncommon and the association of licenses actually does change sometimes. Respectively the keywords "license" and "add" resulted in 75,564 table rows (0.114% of all commits), showing that licenses are also added after the project was already published. Table 4.5 illustrates the results for those keywords in comparison to the years 2012 and 2013.

Table 4.5: Commit messages with keywords on *GitHub*

	2012	**2013**	**2014**
total PushEvents	34,383,307	39,847,469	66,428,295
keywords: license, change	7,110	8,412	11,382
keywords: license, add	57,084	62,105	75,564
keyword: license	106,024	138,905	200,242

After getting a general overview previously, the investigation of several highly starred projects will give additional insights on the association of licenses over time. Therefore, as before, all PushEvents containing the keyword "license" of the following *GitHub* projects were retrieved and the respective commit messages analyzed.

bootstrap:
For *bootstrap* 202 commits containing "license" were found. There are several occurrences of license changes, all from Apache license to MIT license, as well as updates to the Apache-v2.0 license. Apart from that, only some rearrangements of the license file and smaller changes are notable. Running the license analysis on *bootstrap* proved, that for the releases 3.1.1, 3.2.0, 3.3.0, 3.3.6 and 4.0.0-alpha only the MIT license is still in use.

jquery:
For *jquery* 49 commits containing "license" were found. Until version 1.8.1, the commit messages indicate the addition and update of several license files. Then the commit message of version 1.8.1 was published: "license change to MIT only". Since that point, only a few minor changes can be recognized. The usage of the GPL licenses for copiers is explicitly allowed in some commit messages. Investigating the releases 1.12.3, 1.8.1, 2.2.3 and 3.0.0-beta on license usage showed that the MIT license is used in all releases. However, in all of the releases the BSD3 license was found as well. This is caused by some external code used in the project. For the release 1.12.3 those source files are labeled under the BSD3 license. Since release of 1.8.1 those external source files are licenses under BSD3 *or* MIT license, approving the commit message of 1.8.1. In the root folder of all releases, *jquery* states the MIT license.

rails:
For *rails* 40 commits containing "license" were found. The MIT license is used by the ruby on rails project. This license was stated from the beginning, changes can not be recognized, apart from several minor adaptations. This is approved by the license usage analysis for the releases 3.2.22, 4.1.14 and 5.0.0.

docker:
For *docker* 42 commits containing "license" were found. The commits indicate an early switch to WTFPL license and a later switch to the Apache licenses, including an update to Apache-v2.0. They require new additions to apply Apache as well. Several minor adap-

tations can likewise be recognized.

The manual investigation of those four projects approves the previous results. Changes of licenses are not uncommon, but generally the licensing seems stable for most (highly starred) projects. It shows that licensing needs maintenance to some degree, as adaptations for license files or different parts of a project are necessary, especially if different licenses are used, which is probably why *jquery* had its license changed to MIT only. Concluding it can be said, that the association of licenses over time can change, but changes are not trivial. The usage of one license from the beginning is recommendable (see RQ 1.3), although this might be unattainable for most projects.

4.3 License Consistency

Research question 1.3 is about the consistency of licensing in open source projects. To answer this research question, a manual investigation on existing open source projects is accompanied by a literature research, to reveal relevant statistics based on the preceding results. Therefore the licensing of an open source project is investigated on its consistency on two different levels:

1. Is license incompatibility causing inconsistency of licenses?
2. Are changes of the licensing over time causing inconsistency of licenses?

For the first level, a project with only one license is trivially considered consistent. If a project contains more than one license, those have to be compatible to each other, else they are inconsistent. On the second level, the changing of the license of a source file at any time can cause consistency issues, if the files are evolved from the same provenance.

The idea of license compatibility was already explained in section 3.1.5. To investigate the compatibility, the manual investigation is based on the open source projects from *GitHub*, which were utilized in the previous section 4.1 (Run2). Therefore, another ruby script was implemented to automatically retrieve all combinations of licenses, meaning an explicit list of all licenses used per project. This approach clearly showed, that most licenses are used in combination with other licenses. This was to expect, as more licenses were classified than there were projects, in the previous section. Out of 1,167 licenses, only 470 (40.27%) are used as a single license and accordingly trivially consistent. The results

for the most popular licenses can be seen in table 4.6. Combinations of derivations of the same licenses are counted for each license derivation on the single licenses side, so the licenses add up to the total number of classified licenses. For a project's perspective see table 4.7.

Most common are the combinations of Apache (v2.0), MIT and BSD(1-3) licenses. Those three are generally compatible to each other and, given the correct declaration, also combinable. Since single licensing in addition to combinations of those three licenses make the plurality of all used licenses, from this point, 979 licenses are consistent, making 81,01% of the overall licenses. However, there are license combinations with the GPL and BSD4 licenses, which can cause problems, as most GPL licenses contain the *same license* feature, commonly causing compatibility issues, whereas BSD4 generally lacks compatibility with other licenses. This does not necessarily mean they are incompatible, as it has to be considered by case, but there is a certain possibility.

The manual investigation on the GPL and BSD4 licenses combinations shows, that about 51 times, the usage of licenses in combination with other licenses was incompatible the way they are used and therefore inconsistent. Interestingly, all BSD4 license usages turned out to be incompatible with other licenses in that project. This results in a total of 1,116 (95,63%) consistent licenses.

Generally, the license combinations are assumed to be combinable and therefore consistent, which is true for a correct license declaration. Additionally, *other* licenses are not included, the results must therefore be considered as a positive estimate.

Table 4.6: Consistency of licenses on *GitHub*: (Run2)

License	**single license**	**multiple licenses**
MIT	352	231
Apache	70	151
BSD	27	169
GPL	21	146
Sum	470	697

As this calculation leaves out the changes of licenses over time and only considers the current state, the second level still has to be investigated. This aspect was however included by [1]. They investigated the changes in license headers, causing license inconsistencies. In their research, license inconsistency refers to the situation that two source files that evolved from the same provenance but contain different licenses. They state, that 7.2%

Table 4.7: Consistency of licensing of projects on *GitHub*: (Run2)

License	single license	multiple licenses
MIT	352	231
Apache	70	151
BSD	25	120
GPL	15	77
Sum	462	579

of selected file groups of *Debian* reported to have one or more license inconsistencies, mostly caused by license change, meaning two related files contained a different license. In the previous section, the association of licenses over time was investigated as well. Based on those results, the results of [1] can be confirmed. For the highly starred projects, no incompatibility was found, but there were changes of licenses over time within those projects nevertheless, which would be an inconsistency referring to [1]. This however only concerns a fraction of the overall files, supporting the high consistency rate.

Generally, the results show that current open source licensing is mostly consistent. However, license inconsistencies can cause violations and therefore license infringement, or respectively avert further usage. It is important for the developers, to thoroughly check the license if they are reusing code. The next chapter also provides a matrix to examine licenses regarding their compatibility in table 5.3, which can be helpful.

4.4 License Publication

The last sub-question of the first research question covers the publication of a license (RQ 1.4). The way a license is publicized is of major concern for the users of open source software, as well as the owners. The owner has to ensure that the license of his work is properly publicized, as potential users must know about the licensing. The user likewise has the responsibility to examine if an open source project is publicized under a license and follow the regulations. Note, that having no license is also a valid kind of licensing. As well, it is important to note, that different licenses can conflict with other licenses, making the reuse of foreign code to a more complex task and emphasizing a proper publication. Apparently it is of interest for both, the user and the owner, to have a clear communication of the licensing of software.
In the chapter *Related Work*, further background information on legal aspects in publiciz-

ing a license and the importance of a proper license publication were already provided. To get further insights on current publication forms for open source licenses, it is helpful to take a look at existing open source projects and examine them regarding their license publication. Therefore, reference points are given by a preceding literature research and supported by a manual investigation. The manual investigation is inevitable, as new forms of license publication or declaration can not be retrieved automatically. On the basis of those results, advantages and disadvantages of the different forms of publication can be outlined. With those, recommendations or suggestions for a proper publication concept can then be justified.

Due to the research, several ways to properly publicize a license can be identified.

1. Publication with a license file:
   A license file is a file, optimally located in the projects root directory, containing all necessary license information for the project. The most common way is to create a file named ”LICENSE.file” including the whole license text and ideally all authors and the creation date. The file then represents the license for all subdirectories. A weak point of this kind of publication is that projects potentially use foreign code or libraries as well, which could be publicized under a different license. This is probably not expected by a user. However, there are some cases, where this is explicitly noted in the file or additional files.

2. Publication within a file-header:
   Publicizing a license within a license header means, that the license of a project is inserted at the beginning of every single source file in the project in the form of a comment. This kind of publication leaves no space to false interpretations and is the easiest way to ensure a correct licensing in case of re-usage of code oder even code parts. The disadvantage for the developers is the additional effort to license all files manually. The user encounters the same problem when source files contain different licenses. However, some IDE’s facilitate the use of plugins to automatically insert the license. Furthermore, the raw source files increase in their size and tend to look unclear because of a big leading comment section.

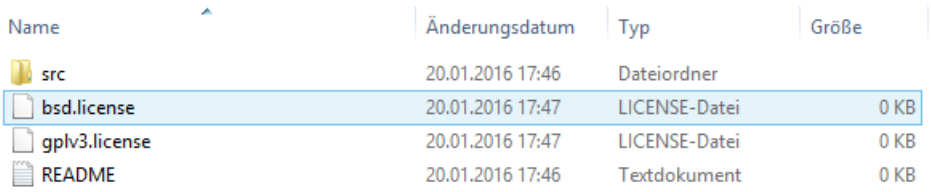
3. Publication via some API:
   Some open source hosts, like *GitHub*[1], provide an API for project licensing. This

[1] https://developer.github.com/v3/licenses/

way it is easy to identify a license of a project by just checking it out from the project site. Also the developer only has to insert the project license once for the whole project. A clear downside however is, that the license publication is dependent on said API, which could cause problems for a potential redistribution, as the license might not be transferred correctly.

There is no solely established way to publicize a license, however, the following approach seems most recommendable. It seems reasonable to combine the advantages of the publication with a license file and the publication within a file-header. Therefore, the license file should contain the license text of the license used in the project and be located at the project's root folder. This is also recommended by the *Free Software Foundation* [21] and even required by some licenses [30]. If there are multiple licenses, extra license files should be added, as the collection of multiple license texts in just one file makes those files really confusing and also the editing can be fault-prone. For eventual automation processes, a common file format for license texts could be an opportunity, an example can be seen in figure 4.6. ".LICENSE" for example, is a file extension which is also supported by different operating systems (e.g. Windows) and should be preferred over ".txt"-files, which are currently the mostly used file format. Additionally every source file should

Figure 4.6: Example for license publication in the root folder



contain a short license notice in its file header, as shown below. This notice is a shortened form of the full license text and is often made available by the licenses providers. Note, that (for example) the actual Apache-v2.0 license has 201 lines of text, the developer would have to put into the header comment, instead of the shortened three lines below.

```
1 /*
2 * This file is published under Apache License (see: apache.license)
3 */
```

By utilizing this way of publicizing a license, all problems would be solved:

1. The license can easily be seen in every file, excluding misunderstandings of the license used, but keeping it short and clear, also without the usage of an IDE. The insertion of the header could be automatized, as it is already common nowadays.

2. All licenses used in the project can be seen in the root folders license files unproblematically.

3. If there are additional licenses from reused code or libraries, they can easily be found by a simple search procedure as well, as they have the common file format "file.license". It would be easy for API's to retrieve the licenses in the same way, increasing their overall value for users. However, currently an own file format for license files is not utilized.

4. This form of declaring a license is compatible with all considered licenses and therefore legally enforcible.

The proposed way of publicizing licenses weak point however is, that it has to be established as a general way, which will be difficult. The currently utilized ways of publicizing a license will therefore be the most prominent approaches in the near future.

Chapter 5

License Recommendation

In this chapter, the possibility of recommending licenses for open source projects (RQ 2) is analyzed. Through the recommendation of licenses, the general usage of open source software, from a developer's perspective, as well as a user's perspective, can be improved. Since there is a distribution of different licenses throughout open source software, an initial conclusion is, that there is some kind of decision-making process in choosing a license and therefore a reasoning behind the choice of a specific license. By taking a look at the different features of licenses, this hypothesis can be supported. The requirement feature *network use* allows users who interact with the software via a network, to have the right to receive a copy of the corresponding source code, meaning that network use is also considered as distribution. Licenses including this requirement are recommendable for software which will commonly be run over a network. This suggests, that projects containing this feature really are indeed running over a network, since the most common license with *network use* is *Affero GPLv3 (AGPL)* and its most prominent alternatives (*GNU GPLv2*, *GNU GPLv3*) include the same features except of *network use*. The distribution of licenses (see chapter 4.1) or [13]) approves, that the usage of the GPL-licenses without *network use* is significantly higher than its alternative, therefore we can infer, that the AGPL-licenses are chosen on purpose because of its additional feature. This kind of reasoning can be done for other license features as well, for example *library usage*. By experience, the usage of license suggestion tools can be approved as well. As [13] verifies, the license usage increased markedly since the introduction of *choosealicense.com*, approving the idea of license recommendation tools.

To finally elaborate a new recommendation or suggestion logic, a methodology is presented initially. Afterwards, the perspective of open source developers on the field of

licensing is further investigated, to get deeper insights on current and typical behaviors. Finally, a recommendation logic is designed, implemented and tested on the realistic use-case of *101companies*.

5.1 Recommendation Logic Design

As the classification of licenses (RQ 2.1) was already covered by literature research in the chapter *Background*, the developer's perspective (RQ 2.2) is the next point of interest. Therefore, several relating ways of recommending a license were described in the chapter *Related Work*. To actually design a recommendation logic methodically, basic information and requirements can be retrieved, by directly addressing developers and investigating their behaviors and perception at first. To further elaborate the design for a recommendation logic, which is an actual improvement to current recommendation tools, already existing tools and logics have to be investigated and possible weak spots have to be detected. The next step is then to analyze if the detected weak spots can be overhauled and how they can be included in a new design. Afterwards, the accordingly created design can be implemented. A following analysis can then evaluate the new recommendation logic on a realistic use-case.

5.1.1 Developer's Perception on Licensing

To gain deeper insights on the perspective of developers towards licensing, which serves as a basis for the next steps, a questionnaire was developed (RQ 2.2). The questionnaire is standardized for every participant, which were selected in online development communities, to assure the correct target group. To participate, a participant has to answer three mandatory questions and is provided a set of answers of which he can choose. Also multiple answers are possible, to facilitate meaningful results (closed-ended multiple choice questions). The idea of the questionnaire is to get a general understanding of user behavior and detect saliences. Therefore, the questionnaire is designed to generate quantitative results, which are comparable among themselves. To reach a high number of participants, the questionnaire is conducted electronically and designed simply. Further surveys based on those questions could extend the questionnaire in the future. The complete published questionnaire can be seen below.
The first question asks, why the participant chose a specific license in his last open source project. The goal of this question is to find out, on which factors the decision-making

of choosing a license was based. The results will be an indicator on the usage of license recommendation and suggestion tools, the general awareness of licenses and the sustainability of the decision-making process, as well as the willingness to deal with the field of licensing.
The second question requires the participant to explain his decision. The results of this question aim to provide a deeper understanding of the thought process behind the license choice. Additionally, it questions the general interest in the field of licensing and the current state of knowledge in the open source development community and therefore also approves the correctness of the first question.
Finally, the third question asks for the license that was actually used in the project. This makes sure that the participant has really published a project and therefore belongs to the target group of the survey. Additionally, it gives another statistic on license distribution as a side effect, which can be used for RQ1.

Questionnaire:

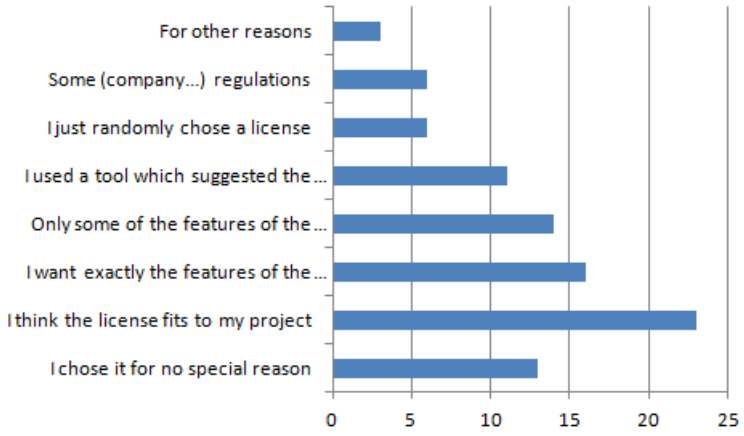
1. Why did you choose the *??* license in your last open source project? Choose all correct answers.
    - I chose it for no special reason
    - I think the license fits to my project
    - I want exactly the features of the chosen license
    - Only some of the features of the license were determining
    - I used a tool which suggested the license
    - I just randomly chose a license
    - Some (company...) regulations
    - For other reasons
2. What do you think of the license you chose? Choose all correct answers.
    - I am well-informed about that license
    - I am sure (no doubt) that I chose the best possible license option
    - There might be a better license choice
    - I do not care which license I chose

- I would like to know more about several licenses
- I would like to know more about licensing

3. Which license did you choose? Insert the license you used.

The surveys results can be seen in figures 5.1, 5.2 and 5.3. Overall, 51 participants were recognized for the survey.
For the first question, the results show, that 45.10% think, that their chosen license also fits to their project. 31.37% assume their choice to be optimal, since they wanted exactly the features, the license provides, whereas 27.45% only consider some of the features as fitting. This shows that there is a clear uncertainty in choosing a license and confirms that there is a need for better license guidance. Other reasons for choosing a license were: license suggestion (21.57%), regulations(11.76%), random choosing (11.76%) and other reasons (5.88%). Especially the results for license suggestions, random choosing and other reasons show, that there is still a lot of potential in recommending licenses. However, the awareness of the importance of properly licensing a project seems rather small.

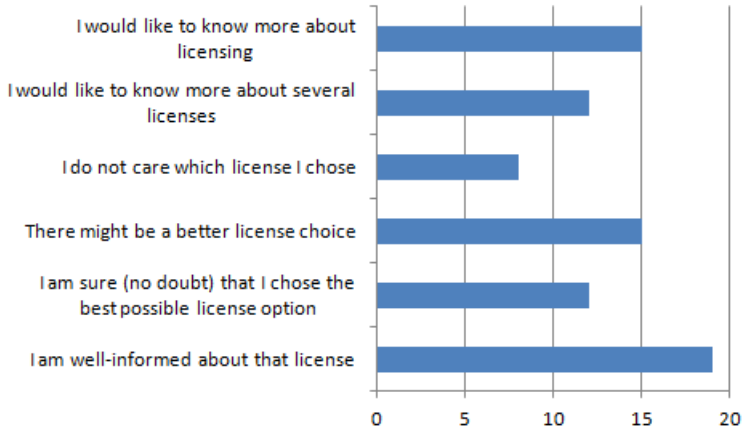
Figure 5.1: Survey answers for the first question



For the second question, 37.25% think they are well-informed about the chosen license and 27.03% are sure of their license choice to be the best possible option, whereas only 23.53% think that there might be a better option. This confirms the assumptions from the first questions, as not even half of the participants consider themselves as well-informed. However, only 23.53% would like to know more about several licenses and 29.41% would like to be better informed about licensing in general, which indicates a lack of interest

in the field of licensing. As 15.69% do not care about their license choice at all, this assumption can be approved. This underlines the importance of a quick and easy form of choosing licenses, as big parts of the open source community do not seem to want to invest a lot of time in it. Those requirements need to be considered when designing a recommendation logic for open source licensing.

Figure 5.2: Survey answers for the second question

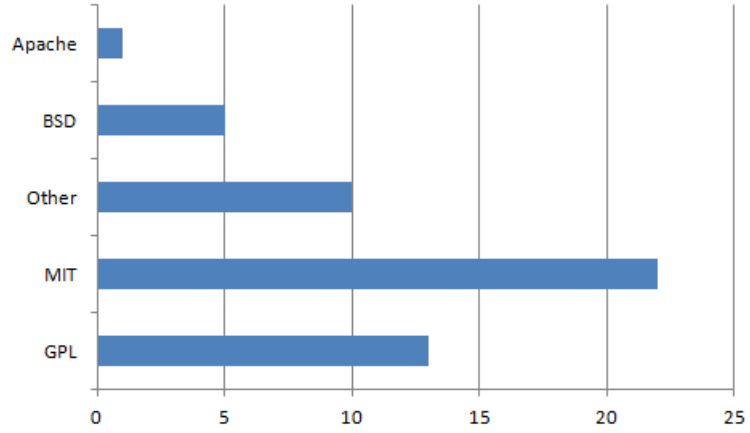


The third question was checkup question. The results however showed, that MIT and GPL licenses are the most popular license choices, followed by BSD and Apache. Other licenses are above average in comparison to the results of the previous chapter. The retrieved trends from the last chapter can however be approved, which also confirms the representativity of the participants of the survey.

5.1.2 Recommendation Logic Requirements

In the course of this thesis many influencing factors on the decision-making of what license to choose were revealed. Conventional tools for suggesting licenses only consider the purpose of the software being licensed. The popularity, the enforcement and different formalities or peculiarities of licenses are not necessarily involved in the process. The following approach tries to pay attention to all influences being explained below. The goal is to not only provide a suggestion for a license choice, but also to recommend licenses, which fit best to the project and its intended usage, by including a ranking-logic that utilizes secondary influences (RQ 2.3). Generally, the recommendation logic needs to be easy and fast to use, but still provide substantial help for the users. Additionally, it needs

Figure 5.3: Survey answers for the third question



to provide extra value in comparison to conventional logics. As users should be able to rely on the logic, it has to be unfailing and trustworthy. As well, licenses not only need to be included in the largest possible number, but also completely and legally correct. Therefore, an implementation should support the addition of new or editing of existing licenses or knowledge. Possibly providing it as open source software will encourage a growing knowledge base. Other requirements can be directly derived from the following design approach.

To suggest a license, the first guiding principle should always be, what the later license holder's (developer's) general intention is. Therefore, the features (requirements, permissions, constraints) of licenses can be used as an indicator. By selecting the most important features in the view of the developer, licenses with conflicting features can be excluded from the list of possibly appropriate licenses. There is a possibility, that there is no perfectly matching license for the developers desires, so suggestions and possible problematics must be pointed out during this process interactively, to facilitate a reaction on the feature choices.
Optimally, the previous step leads to a list of possibly appropriate licenses. To provide further recommendation, also the following secondary influences can be considered.

1. Popularity: A popular license has several advantages in comparison to less popular license. Popular licenses are not only used more, they are also better known by the community. Therefore, more people know what the license is about and it is more likely that they know how to properly use it. The most popular licenses

also tend to have associations backing them up and a user can be sure of their juridical correctness. The popularity of a license is also an important factor to its enforcement. [30] also recommend the use of “formal” licenses, which the most popular licenses all are.

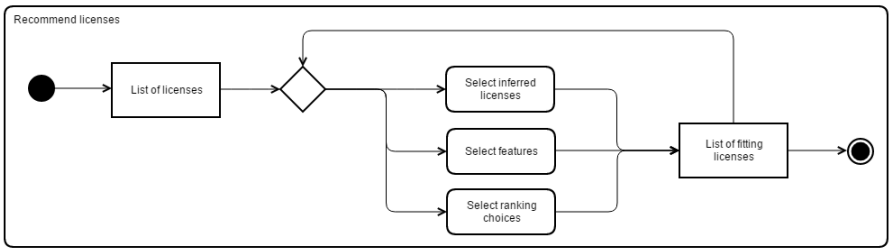
2. Enforcement: The enforcement of licenses is a difficult field. As previously described, there have been some cases of license infringement. It is to assume that licenses, which have already proven their validity in court, are relatively safe to use. Licenses with an approving verdict or leading case are therefore preferable.

3. Compatibility: Not all licenses are compatible to each other. If a user has already inferred licenses in his code, probably through foreign code or libraries, incompatible licenses have to be excluded from the list. This should be done preliminary. Exemplary techniques (scripts) to analyze a project regarding currently used licenses and license combinations were presented in the previous chapter.

4. Combination of licenses: If several different licenses are used in a project, it is not always clear, which license is the overlapping one. Although licenses are compatible with each other, some require to be declared on the highest level of licenses and will be determining.

5. Deprecated licenses: Licenses are updated from time to time. Although the features of a license remain nearly identical, special or problematic clauses may influence the recommendation of that license and are eliminated in newer versions. Therefore, newer versions might be a better choice.

Figure 5.4: UML activity diagram of the recommendation logic



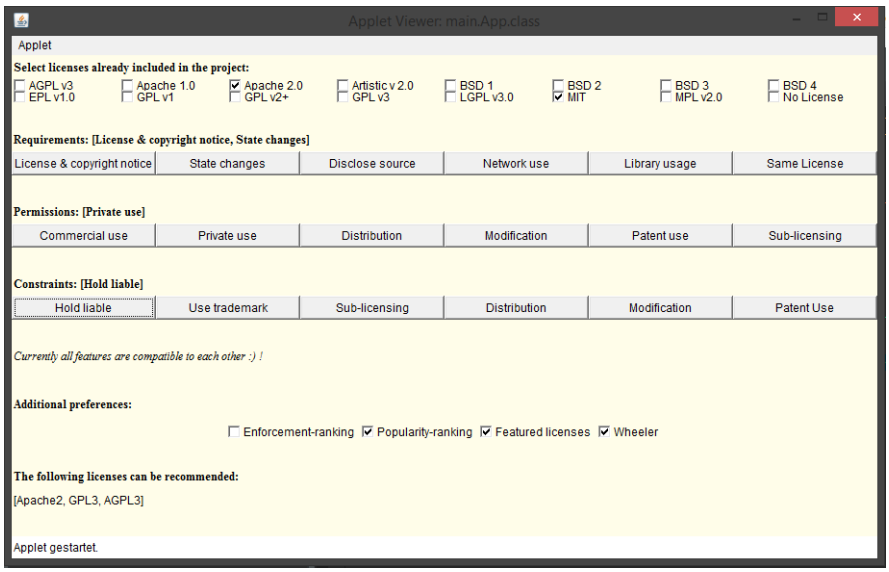
Respective to those requirements, a possible recommendation logic could be designed like the UML activity diagram in figure 5.4 models. The first input for the recommendation logic is the user’s intention, which is expressed by the user selecting license features

in a selection menu, that he denotes useful. Those features are interactively checked regarding their compatibility with already selected features. An additional input are the licenses which are already used within the project, e.g. by different libraries, to ensure the compatibility of those licenses and the currently suggested ones. A license which is in conflict with already used licenses, can neither be suggested nor recommended. With the collected input data, a list of all possible licenses can be created. To provide further recommendation, the user can select a method of ranking the suggested licenses, for example a ranking via license popularity or license enforcement, or additional preferences. As the ranking rates the different licenses for the special use-case, in the end, the user has a well-considered license recommendation.

5.2 Recommendation Logic Implementation

The previously proposed logic design was implemented as an *Java* applet, to provide an alike online and offline solution for the tool. This provides an easy-to-use graphical user interface with different checkboxes, buttons and several labels (text-fields), which are updated on action. The graphical user interface of the license recommendation applet can be seen in figure 5.5. Although the design has to be adapted to modern standards and more licenses need to be added, the logic is fully implemented and working. It can easily be integrated in a website as well.

Figure 5.5: License recommendation applet (v1.0)



The *License Recommender* consists of three classes. *Licenses.java* provides all included licenses and their features. The features of several licenses can be seen in appendix table 1. *App.java* contains the main method and the applet logic. And *LicenseSuggestion.java* provides the recommendation logic.
The following code snippet contains the method *SuggestLicense()* from the file *LicenseSuggestion.java*, basically showing how the recommendation of licenses works in the applet.

SuggestLicense() (shortened):

```
1  public LinkedList<String> SuggestLicense(){
2  LinkedList<String> suggestedLicenses = new LinkedList<String>();
3  suggestedLicenses = CheckFeatures();
4  suggestedLicenses = CheckInferredLicenses(suggestedLicenses);
5  if(!isCheckedA() && isCheckedB()){
6  //include popularity ranking
7  }
8  if(isCheckedA() && !isCheckedB()){
9  //include enforcement ranking
10 }
11 if(isCheckedA() && isCheckedB()){
12 //include popularity- and enforcement ranking
13 }
14 if(isCheckedC()){
15 //remove licenses which are not featured
16 }
17 if(isCheckedD()){
18 //include the wheeler license slide
19 }
20 return suggestedLicenses;
21 }
```

At first, a new and empty *LinkedList* of strings is created, which later contains all the recommended licenses in a ranked order. By calling the method *CheckFeatures*, the features of each license are compared with the features the user selected, returning all respective licenses. This works by comparing the predefined *LinkedList*'s for every license, with the user's input. Every License therefore has a list of permissions, requirements, constraints and incompatible licenses. As the result list is updated interactively, unwanted features can be tested, to see which licenses are not desired by the user and deselected afterwards. Therefore, all buttons and checkboxes are implemented with a toggle function. The next line of *SuggestLicense()* uses the incompatibility list and checks if the licenses are com-

patible with the already used (inferred) licenses. The following if clauses refer to the optional preferences and ranking choices. The ranking rules are based on this thesis results. The popularity is implemented according to the second run of the previous chapter. A license is labeled as *enforcible*, if a verdict has proven its enforceability (see table 5.1). The compatibility of features and licenses is expressed in tables 5.3 and 5.4. The licenses compatibility table (5.3) also includes special clauses of licenses which are not necessarily expressed as features and the combinability of licenses. If both are selected, popularity is preferred over enforcement as a standard, as enforcement data is still a growing field in the open source sector. Therefore the most used licenses are on top and the lesser used licenses (less than 1%) are sorted by their enforcement. All steps are repeated on every action (something changes) of the user.

Table 5.1: Enforced licenses (verdict)

License(s):	**verdict**	**Source**
GPL (v3, v2)	supported	[31], [32]
LGPL	supported	[32]
GPL (alternatives)	likely	[32]
BSD	likely	[15]
MPL	likely	[15]
Ohters	unknown	

The additional choice *featured licenses* deletes all licenses from the list, which are not featured, those licenses can be seen in table 5.2. It leaves out licenses like the Apache-v1.0, as the *Apache foundation* recommends the usage of the newer version. Another choice is to include the *Wheeler license slide* [7], checking which license is to declare, for combinations of licenses. Those preferences are selected by default, as they are recommended for the user. The full code of the *LicenseRecommender* will also be attached to this thesis.

Table 5.2: Not recommendable licenses

License:	**Reason:**
Apache-v1	Deprecated version of Apache-v2
BSD4	Lack of compatibility because of special clauses
GPLv1	Deprecated version of GPLv2 or GPLv3
Artistic 2.0	Not yet included in the license slide
Eclipse Public License	Not yet included in the license slide
No license	All positive effects of open source software are reversed

Table 5.3: Compatible and combinable licenses rules (shortened)

Licenses:	MIT	BSD4	BSD3	BSD2	BSD1	Apache v2	Apache v1	GPLv3+	GPLv2+	GPLv1	AGPLv3
MIT	X	X	X	X	X	X	X	X	X	X	X
BSD4	-	X	-	-	-	-	-	-	-	-	-
BSD3	-	X	X	X	X	X	X	X	X	X	X
BSD2	-	X	X	X	X	X	X	X	X	X	X
BSD1	-	X	X	X	X	X	X	X	X	X	X
Apache-2	-	-	-	-	-	X	X	X	-	-	X
Apache-1	-	-	-	-	-	-	X	-	-	-	-
GPLv3+	-	-	-	-	-	-	-	X	-	-	X
GPLv2+	-	-	-	-	-	-	-	X	X	-	X
GPLv1	-	-	-	-	-	-	-	X	X	X	X
AGPLv3	-	-	-	-	-	-	-	-	-	-	X

This table was created with great care, however, reading the original sources is recommended. Additional compatibility overviews of (other) licenses can be found there as well: [7], [30], [33], [34], [42]).

How to read: The licenses column is the license already used in the project, the licenses row depicts the license to be added as the determining license (includes combinability). For compatibility only, read in both directions and check if there is at least one X.

Key overview:
X : Licenses are compatible (combinable)
- : Licenses are incompatible (incombinable)

5.3 Recommendation Logic Appliance

In the previous sections, a license recommendation logic was designed and implemented. Using the example of the *101companies* project, the logic can be tested and further evaluated (RQ 2.4). The *101companies* project is a good test-case, as its complex structure and its versatile features involve all the different parts of the logic.

Table 5.4: Contradicting features of licenses rules

Constraints: **Permissions:**	Modification	Distribution	Sub-Licensing	Patent use
Modification	X			
Distribution		X		
Sub-Licensing			X	
Patent use				X

5.3.1 Use-Case Requirement Analysis

To enable the evaluation of the later solution for the *101companies* project (and therefore the recommendation logic), all requirements on a licensing concept for the project have to be determined. General information of the *101companies* project were therefore given in the chapter *Background*, identifying stakeholders and other general aspects. The following list specifies all requirements and processes for each step of the recommendation. This list is meant to be applicable to other projects as well, but tailored for *101companies*. All steps can be executed with the tools presented during this thesis. The meta model in figure 5.6 shows how the licensing is meant to be conceptualized. The project itself is licensed under one license, whereas each contribution may have another license (e.g. in used libraries), compatible with the projects license. This way, the *101companies* project is licensed as one project, including all contributions as sub-projects.

1. **General Requirements**

   (a) The licensing of the project must be coherent.

   (b) The license for the project must be suitable (academic project with many stakeholders).

(c) The licensing concept should be extensible, as the *101companies* project is still growing.

(d) Every part of the project must be licensed (excluding no-license-licensing).

2. **Prerequisites:**

   (a) All currently used licenses must be included (license usage analysis).

   (b) All currently existing license combinations must be compatible, to ensure legal and enforcible licensing

   (c) In case: all incompatible licenses must be eliminated.

3. **Feature Consideration**

   (a) All requirement features must be considered.

   (b) All permission features must be considered.

   (c) All constraint features must be considered.

4. **Evaluation of license options**

   (a) All preferences must be considered (Wheeler, featured licenses).

   (b) All ranking options must be considered.

5. **License publication:**

   (a) The license declaration must be legally correct.

   (b) Contributions can be licensed automatically.

5.3.2 Use-Case Requirement Realization

As shown in the requirement specification before, the first step is to check which licenses are already used in the contributions of *101companies*, by applying a license usage analysis. Therefore, the techniques of the previous chapter can be utilized. The results of the analysis for all 228 contributions can be seen in table 5.5.
Table 5.6 shows in which combinations those licenses occur and if those license combinations are compatible. Incompatible combinations would have to be excluded from a prospective licensing concept, as they are not legally enforcible. Luckily, a manual

Figure 5.6: Meta model of a *101companies* licensing concept

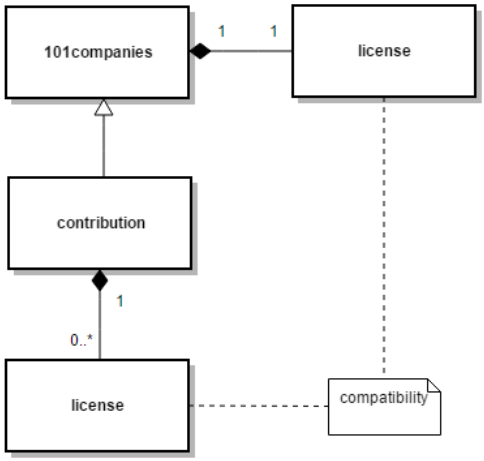


Table 5.5: Licenses identified in the *101companies* contributions

License	**Distribution [abs]**
Apache 2.0	4
spdxBSD3	1
BSD3	2
BeerWare	1
GPL-2.0	1
GPL-2.0+	6
GPL-3.0+	1
LesserGPL	4
MIT	3

Table 5.6: License combinations identified in the *101companies* contributions

Combination	**Compatible**
MIT	yes
Apache 2.0	yes
GPLv2	yes
BSD3, MIT	yes
Apache 2.0, GPLv2, LesserGPL	yes
BeerWare, GPLv2	yes
GPLv3, LesserGPL	yes
BSD3, GPLv2, LesserGPL	yes

investigation showed, that all combinations are compatible within the *101companies* contributions, so no license must be eliminated.
The next step is the consideration of features. This step is postponed at this point, as the results of the previous step generate a special case: only one license can be suggested in regard of the current information. Selecting different features would not extend the results. However, with this information, different licensing options can be constructed. Those will be presented and discussed below.

Option 1:
If the preferences *Wheeler* and *featured Licenses* are included, which they should be, the Affero GPL v3.0 License is the only recommendable license. The reason for that is, that already many different licenses are included in the contributions and therefore limit the options. Especially the GPL licenses *same license* feature is a reason for that, which is often criticized [35]. However, the AGPL license would be a possible comprehensive license for the *101companies* project. It is to note, that the AGPL license is a strongly protective license and therefore lacks compatibility with other licenses, once it is used within the project. Several contributions could be licensed differently on a lower level, but the AGPL license will outweigh them. However, software with other licenses could be included nevertheless, under those circumstances.

Option 2:
Derived from the first option, it could be a promising approach to eliminate licenses, which are already used in the project. This is not inevitable, but the elimination of licenses would have the consequences, as illustrated in table 5.7 and provide lots of different licensing options. Especially the elimination of the protective licenses would enable a lot of additional possibilities to license *101companies*, particularly the permissive licenses. Those could be advantageous for an academic project with many different contributors, as it eases reuse and collaboration. The selection of desired features in the recommendation tool could then provide new license options or features, for example features like *same license* could be excluded, making derivative work easier. The second step of the recommendation process could then be applied meaningfully. Generally, the following features seem reasonable for the *101companies* project:

1. Requirement *state changes*, to easily evaluate code changes.

2. Permission *private use*, as it is an academic project which is also supporting learn-

ers.

3. Permission *modification*, to allow modifying contents.
4. Permission *distribution*, to allow the distribution of contents.
5. Constraint *hold liable*, as there is no warranty on the functioning of code.

And the following features should probably not be included:

1. Requirement *same license*, as explained before.
2. Permission *patent use*, as the patent should remain with the distributor.
3. All contradicting constraints: *modification* and *distribution*.

The selection of this feature combination supports the usage of the Apache-v2.0 license. If *state changes* is however not explicitly included, also MIT, BSD(1-3) and MPLv2 can be recommended.

Table 5.7: Consequences of license elimination for the *101companies* contributions

Eliminated licenses	**Additional options**
GPLv2	GPLv3
GPLv2, GPLv3	GPLv3, LGPLv3
GPLv2, GPLv3, LGPLv3	GPLv3, LGPLv3, Apache v2
GPLv2, GPLv3, LGPLv3, Apachev2	GPLv3, LGPLv3, Apachev2, BSD(1..3), GPLv2, MPLv2
GPLv2, GPLv3, LGPLv3, Apachev2	GPLv3, LGPLv3, Apachev2, BSD(1..3), GPLv2, MPLv2
GPLv2, GPLv3, LGPLv3, Apachev2, BSD3	GPLv3, LGPLv3, Apachev2, BSD(1..3), GPLv2, MPLv2, MIT

Option 3:
The third option is to choose a different licensing approach and license all contributions individually. An advantage would be, that the licenses could be adapted to the requirements of each contribution, especially if certain features are desired. As well, the inclusion of new contributions that already contain a license would be easy. However, the coherency requirement would be failed. There would be a multitude of different licenses throughout

the *101companies* project, promoting an unclear and confusing licensing concept. This concept is currently utilized, by not specifying a *101companies*-license. However, most contributions are currently licensed under no license, which should be changed then, to enable proper open source usage.

5.3.3 Use-Case Solution

Previously, three possible and valid licensing options were presented. All of those options have different advantages and disadvantages. It seems recommendable, to firstly try the second option, as a minimum of included licenses in a project always makes reuse and maintenance easier and permissive licenses seem more suitable for academic projects. The first option can still be used as a fallback option afterwards, if license elimination is not possible or the AGPL license is still desired. The third option does not meet all requirements, therefore it is not recommendable, although it is a valid form of licensing. Independent from which approach is finally desired, it is possible to find a licensing strategy for a complex project like *101companies*, proving the value of the recommendation logic as well as its usability.
The next step is to declare the license within the project. Therefore, it is recommended to use a short license header and a license file in the top folder, as explained previously. Afterwards the project is successfully licensed. Regardless of the desired option, it is suggestive to eliminate other licenses step by step in the future, to minimize maintenance and potential conflicts. Also the inclusion of new (not already used licenses) should be avoided. If some of the contributions shall however be licensed differently than the comprehensive and coherent license, those have to be investigated separately case by case.
Nevertheless, some limitations have to be outlined. Several licenses in the *101companies* contributions are not included in the recommendation logic yet (BeerWare), hence, they cannot be checked regarding their compatibility with other licenses, which had to be done manually. Equally, maybe more licenses are possibly recommendable. Additionally, the recommendation process was conducted with background knowledge of the tool and the licenses. It must be assured, that users of the tool are provided all necessary information. For this use-case, the results should however satisfy, as a comprehensible solution is found.

5.4 Recommendation Logic Evaluation

In the chapter *Related Work* the license recommendation tool choosealicense.com was presented. Now, that a new recommendation logic was implemented and tested in the course of this thesis, it is to analyze, if there is a surplus value in the new idea.
Choosealicense.com has its advantages in its simplicity. The user just has to read the three slogans and can choose one of the featured licenses directly, which is a very fast and easy method to choose a license. If the user wants a wider spectrum of choices, choosealicense.com provides a clear and well-arranged overview over additional licenses.
However, there are disadvantages which the suggested recommendation logic tries to avoid. By selecting the fundamental features of a license, the user can investigate a large amount of different licenses in a reactive manner. This way the user should get a much deeper understanding of the license for his own software and misconceptions can more likely be avoided. As well, the list of considered licenses is much longer than in other recommendation tools, so there is a chance of choosing a license that actually fits better to the project. With the idea of providing different choices of ranking, the suggested licenses can help the user in the choosing process additionally and provide a recommendation. Since the compatibility and the combinability of licenses is included as well, clearly wrong decisions can be excluded from the process. Therefore, also the provided techniques for analyzing an existing project can be helpful.
A disadvantage of the new approach is that it takes a little more effort from the user, who needs to inform himself about the different features. This could probably discourage some users from using the tool.
Concluding it can be said, that the suggested recommendation logic is something in between the minimalistic approach from choosealicense.com and reading the license text of all licenses self-initiated. It seems that the advantages outweigh the disadvantages, however a final evaluation needs to include the developers and authors themselves. Therefore, the logic was applied on the *101companies* project in the last section. It was able to recommend a license, which is fitting for the project and provides a proper form of licensing. Therefore, the proposed recommendation logic proved itself as a good assistance in licensing complex projects. In comparison, a manual investigation of the project with such a great number of contributions would have been an enormous and time-consuming effort. With a certain amount of background knowledge, the license recommendation tool can generate and provide good results.

Chapter 6

Concluding Remarks

The following chapter will summarize the thesis results and give an overview over limitations and the consequential future work. Some closing words will round off the elaboration of this thesis.

6.1 Summary

In the chapter *Introduction*, two research questions were presented and answered in the subsequent chapters. The results can be shortly summarized the following:

RQ 1: What is the state-of-the-art in licensing open source software?
To answer this question, several sub-questions were designed. Each of them contains different statistics and information. Together, they represent the state of the art in open source licensing.
RQ 1.1: What is the distribution of licenses in the open source sector?
The distribution of licenses was acquired by an automated approach for *GitHub*. Several runs of analysis yield, that the MIT license is the dominating license throughout the open source sector. The GPL, BSD and Apache licenses are less popular but still widely distributed. It seems that the more permissive licenses are currently in favor of the open source community.
RQ 1.2: Are there changes in the association of licenses over time?
The investigation showed that changes in the association of licenses over a period of time happen. Those changes are not trivial and can cause several difficulties, which have to be

considered by the developers (license holder).

RQ 1.3: How consistent is the usage of licenses?

The results showed that current open source licensing is mostly consistent. However, it is common that licenses are used in combination with other licenses, therefore license conflicts need to be considered. It becomes apparent that there is a certain amount of projects, which are not correctly licensed and may hence cause violations of licenses.

RQ 1.4: How can licenses be publicized properly?

The most common way to publish a license is a license header as comment in the beginning of every source file. In addition, the integration of a license file for every used license, containing the full license text is common. This is also the most recommendable way to publish a license. Licenses API's can however be a sensible addition.

RQ 2: Can we provide a recommendation logic for licenses?

This question was split in several sub-questions as well. By answering those questions step by step, a recommendation logic was designed, which proves the possibility of designing one.

RQ 2.1: How can licenses be classified?

There are many ways of how to classify licenses on different levels. Various approaches were outlined throughout this thesis. The most prominent way is the differentiation between several levels of permissiveness of a license.

RQ 2.2: What is the developer's perspective on the field of licensing and which inferences regarding license recommendation can be made?

The results showed that the process of choosing a license is uncertain for many developers. It is clear that there is a need for a better license guidance and potential for recommendation and suggestion logics.

RQ 2.3: How can licenses be suggested or recommended?

To recommend licenses, a recommendation logic was designed and implemented. It works by utilizing several levels of information, retrieved through the user of the tool and different knowledge bases. This way the user's intention, as well as given restrictions are considered and combined.

RQ 2.4: How can a recommendation logic apply to complex open source projects like the *101companies* project?

The recommendation logic worked well for the *101companies* project. It is possible to recommend a license for projects with many contributions containing different licenses with the proposed approach. Different licensing options could be provided.

6.2 Limitations and Future Work

The results for the thesis research questions were summarized before, however, some limitations have to be mentioned. Those limitations provide different opportunities for future work.

6.2.1 License Usage Analysis

The limitations of the chosen approach of identifying the distribution of licenses were already listed in the chapter *License Usage Analysis*: the accuracy of *Ninka* and the inaccuracy in the statistic of projects that do not have a license. An additional point of limitation is the number of investigated projects. As the computing effort for running the analysis and the amount of data was enormous, as [2] can approve, it is a particularly time-consuming process to analyze open source projects on a large-scaled basis. The 1,330 investigated projects of the first run contained about 29.7 gigabytes of data distributed in 100,234 folders containing 5,000,181 files, although the big and irrelevant data was previously deleted. Especially the amount of single files, which have to be processed, is a complex task. With more computing power and a higher network bandwidth the results could be enhanced and more precisely. As a side note, the analysis with *Ninka* was performed on an Ubuntu 14.04 virtual machine with 2GB DDR2 Ram and two CPU cores from a *Intel Core2 Quad Processor Q6600*. Other tasks on a second virtual machine with 4 gigabytes DDR3 Ram and one core from an *Intel Core i5-4300U*. For the other sub questions of the first research question, the limitation of the number of investigated projects must be considered as well, as more input data always creates more representative work. Additionally, any suggestions, like the idea of how to publish licenses, are dependent on the open source community.

This leads to the following opportunities of future work. In general, the analysis of the distribution of licenses can be expanded on a higher number of projects to improve the results. An additional approach for the license usage analysis would be, to further differentiate between the types of projects that are analyzed and specify criteria, as in the second run of this approach. In this research, the analyzed projects were chosen from the *GitHub* search API, it could be interesting to see, if and which licenses are dominant, for example for a single programming language, and if those results differ from the overall results, just as the second run of the thesis already indicates. As mentioned before, also the retrieved statistics will be out-dated sometime and therefore not be useful anymore,

unless they are updated. Generally, as time progresses, all retrieved statistics for the first research question might be in need of an update, to identify the latest trends.

6.2.2 License Recommendation

The idea of creating a license recommendation logic is relatively young and currently there is only one dominating license suggesting logic powered by *GitHub*. In this thesis, a new approach was presented, which might be an improvement to known tools, providing a logical recommendation. However, this approach is limited to the user acceptance and needs to be featured by larger instances to get any attention. Otherwise, the idea of spreading the knowledge for the proper usage of licenses cannot be supported with it. However, the survey generally showed a certain interest. Like all recommendation logics, also the new approach is limited to a set of licenses, which are included.
As a side note, especially the enforcement, combinability and compatibility of licenses is a field for jurists, computer scientists can only make suggestions and expose problems. As of now, it seems, that the enforcement of open source licenses is not a common topic for law-firms, so the state of the art is a bit unclear. The compatibility (including combinability) of licenses must therefore also undergo professional legal investigation. Those results have to be included in the tool.
The approach itself can still be improved by giving the tool a user friendlier design and by constantly adding licenses and further data relevant for the ranking. As well, the tool is not tested by a large group of users, so feedback might bring up some further limitations. The test with the *101companies* project however was promising.
As a consequence from said limitations, the license recommendation applet needs a certain amount of maintenance. Also, it is yet to publish for public access, therefore, a website or alternate ideas have to be implemented. This platform also needs to deliver the necessary background knowledge for a proper utilization of the logic.

6.3 Closing Words

The thesis *License Usage Analysis and License Recommendation in Open Source Software Development* provides various insights in open source software development and licensing. Those are the basis for this section, commenting on several general aspects and questions, as well as a view on certain trends in open source development regarding

licensing, to provide a perspective on the results. As an introduction, the general idea behind open source software can be questioned.

There are many reasons why open source software is good for companies, developers and software users and the utilization is well-marked in many different areas. Open source software keeps costs down, improves quality, delivers business agility and mitigates business risk [36]. Additionally, it grants freedom, flexibility and is non-binding [37]. Open source software archives provide easily accessible, mostly high-quality and versatile software that can be used by anyone and generally without any charging. Private users, as well as professionals, are the profiteers, both in a private and a working environment. So as apparently the users are the ones who profit most from open source software, what is their role?

Since the usage of open source software obliges the users only to few conditions, which are rarely enforced, those rules or some etiquette should be followed. The least thing a user of open software can do, is to give credit to the owner, for providing his work. If open source software is a key for you earning a lot of money though, especially as a company, a donation or sign of gratitude should really be considered. Nevertheless, the compliance with those ideas seems rather small, so what can the developers do?

The developers of open source software can be private persons, companies, as well as free development groups. It is them, who put the work into the software, mostly without any payment. If they want to get something back for their work (give and get), they have to question their own intentions. So what are the intentions behind open source programming? Zed Shaw writes in [38]: “I wrote Mongrel and then gave it away, on the hopes that it would help a bunch of other people, and that giving it away would come back to me in some way. Maybe a job, or some respect, or hell maybe my own company doing more software like it”. Those are probably the most common intentions behind open source development. However, Shaw did not get anything back, although the software was a huge success. As his, as well as general intentions seem rather naive, relying on trust and decency, open source needs software licenses and not only software licenses, but fitting software licenses, to enforce developer’s intention.

The best way to get an understanding of which the best license for a certain project is, is the reading of the licenses texts. As this is a time-consuming task, license recommendation or suggestion tools can help. In the course of this thesis, one tool was presented and one developed, specially designed for unexperienced developers. Of course also those tools might not be sufficient for everyone. Nevertheless, it is important to make a decision and be aware of the consequences. The case of Shaw can be exemplary here as well, as he

learned from his earlier mistake. He now prefers to use the GPL-licenses, to force users of his software, especially companies, to admit the actual usage [38], which is perfectly comprehensible and emphasizes the importance of choosing a fitting license. Yet, Shaw reports of complaints of his choice of license, as it makes things harder for some of the users [39]. This impression approves, that some people do not cherish open source software enough, as it should be and even complain about freely accessible software, although the *naive approach* clearly does not work. Therefore, all different licenses should be accepted in the open source community, since they are used by the developers on purpose. In addition, the common knowledge and awareness should be improved, as open source software development is a promising advancement for the IT sector.
In the end, every developer, as professional in a company or in private, can benefit from sharing code, so the least that should be done, is to accept the conditions of usage and give the developer the earned credit without exception.

Appendices

Licenses and their Features

The following table provides a large collection of open source licenses and their features. The list might not be exhaustive and no warranties can be made, although the list has been carefully crafted. Fore more information please refer to the license text of each license.

The following labels are used for the different features:
Requirements: License and copyright notice (R1), State changes (R2), Disclose source (R3), Network use (R4), Library usage (R5), Same License (R6).
Permissions: Commercial use (P1), Private use (P2), Distribution (P3), Modification (P4), Patent use (P5), Sub-licensing (P6).
Constraints: Hold liable (C1), Use trademark (C2), Sub-licensing (C3), Distribution (C4), Modification (C5), Patent use (C6).

Table 1: Lookup-table for licenses and their features

License	**Requirements**	**Permissions**	**Constraints**
Apache License 2.0	R1, R2	P1, P2, P3, P4, P5	C1, C2
GNU GPLv3	R1, R2, R3, R6	P1, P2, P3, P4, P5	C1
GNU GPLv2	R1, R2, R3, R6	P1, P2, P3, P4, P5	C1
Affero GPLv3	R1, R2, R3, R4, R6	P1, P2, P3, P4, P5	C1, C3
MIT License	R1	P1, P2, P3, P4	C1
Artistic License 2.0	R1, R2	P1, P2, P3, P4, P5	C1, C2
Eclipse Public Licnese 1.0	R1, R3,R6	P1, P2, P3, P4, P5	C1
(BSD) ISC License	R1	P1, P2, P3, P4	C1
BSD 3-Clause	R1	P1, P2, P3, P4	C1
BSD 2-Clause	R1	P1, P2, P3, P4	C1
GNU LGPLv3	R1, R3, R6	P1, P2, P3, P4, P5	C1
GNU LGPLv2.1	R1, R3, R6	P1, P2, P3, P4, P5	C1
Mozilla Public License 2.0	R1, R3, R6	P1, P2, P3, P4, P5	C1, C2
(Public Domain) The Unlicense		P1, P2, P3, P4	C1
(Public Domain) CC0 1.0 Universal		P1, P2, P3, P4	C1, C2, C6

Classified Licenses: License Usage Analysis

AAL
AFL-1.1, AFL-1.2, AFL-2.0, AFL-2.1, AFL-3.0
AGPL-1.0, AGPLv3+
Apache-1.0, Apache-1.1, Apache-2.0
ArtisticLicensev1
BeerWare
boost-1
BSD1, BSD2, BSD3, BSD4, BSD-4-Clause-UC, spdxBSD2, spdxBSD3, spdxBSD4, BSD-Doc, BSD-style
CC
Cecill
CPL
DoWhatTheFuckYouWant
emacsLic
EUPL, EUPL-1.0, EUPL-1.1
FSFUnlimited
GFDL-1.1, GFDL-1.2, GFDL-1.3
GPLv1, GPLv1+, GPLv2, GPLv2+, GPLv3, GPLv3+
IBMv1
Intel
MPLv1
kerberos
LesserGPL
LGPL, LGPL-2.1, LGPL-2.1+, LGPL-2.1+-KDE-exception, LGPL-3.0, LGPL-3.0+
LibraryGPL
MIT, MIT-style
MX4J
NPLv1
OSL-1.0, OSL-1.1, OSL-2.0, OSL-2.1, OSL-3.0
phpLic
PLv2
postgresql, postgresqlRef

public domain
QPLv1
SameAsPerl
simpleLic2, simpleLicense1
sunRPC, SunSimpleLic
W3CLic
X11CMU
zendv2

License Distribution Analysis Data

Table 2: Analysis results of the total distribution of all classified licenses on *GitHub* (Run1)

License	**Distribution [abs]**	**Distribution [%]**
MIT	599	34.79
Apache	282	16.38
spdxBSD3	141	8.19
GPL-2.0+	135	7.84
GPL-3.0+	115	6.69
spdxBSD2	88	4.48
LesserGPL	88	5.11
GPL-2.0	63	3.66
BSD3	37	0.15
LibraryGPL	24	1.34
LGPL	18	1.05
MPLv1	16	0.93
boost-1	14	0.81
spdxBSD4	14	0.81
Intel	14	0.81
FSFUnlimitet	13	0.75
BeerWare	9	0.52
simpleLicense1	7	0.41
SameAsPerl	7	0.41
X11CMU	5	0.29
BSD2	4	0.23
LGPL-3.0	5	0.29
kerberos	3	0.17
Apache-1.0	3	0.17
SunSimpleLic	3	0.17
sunRPC	3	0.17
doWhatTheFuckYouWant	3	0.17
BSD4	2	0.12
GPL-1.0	2	0.12
GPL-1.0+	2	0.12
GPL-3.0	2	0.12
NPLv1	1	0.06

Table 3: Analysis results of the total distribution of all classified licenses on *GitHub* (Run2 (highly rated projects))

License	**Distribution [abs]**	**Distribution [%]**
MIT	583	48.34
Apache	213	17.66
BSD3	24	1.99
spdxBSD3	94	7.79
spdxBSD2	65	5.39
GPL-2.0+	51	4.23
GPL-3.0+	47	3.99
LesserGPL	28	2.32
GPL-3.0	16	1.33
GPL-2.0	13	1.08
Intel	12	1.00
Apache-1.0	7	0.58
LibraryGPL	7	0.58
FSFUnlimitet	5	0.41
LGPL	5	0.41
spdxBSD4	5	0.41
BSD4	4	0.33
BeerWare	4	0.33
BSD2	4	0.33
boost-1	4	0.33
MPLv1	4	0.33
doWhatTheFuckYouWant	2	0.17
SunSimpleLic	2	0.17
Apache-1.1	1	0.08
CC	1	0.08
NPLv1	1	0.08
phpLicense	1	0.08
public domain	1	0.08
SameAsPerl	1	0.08
w3cLic	1	0.08

Additional Implementation Excerpts

Main.defines.java of the GitDownloader program (shortened):

```
1  /* File for the search results */
2  public static final String searchResultFile = "src/RetrievedData/Search_results
       .txt";
3
4  /* keyword to filter the search results: default = "full_name" */
5  public static final String filterKey = "full_name";
6
7  /* HashSet size in removeDuplicates: default = 10000 */
8  public static final int removeDuplicatesHashSet = 10000;
9
10 /* how many pages of the GitHub search are crawled: limit = 35 */
11 public static final int maxSearchPages = 35;
12
13 /* requests of the crawler until timeout */
14 public static final int maxCrawlerRequests = 8;
15
16 /* duration of timeout before continuing to request: default = 60000 */
17 public static final int crawlerTimeout = 60000;
18
19 /* general GitHub API search link */
20 public final static String gitSearch = "https://api.github.com/search/
       repositories?q=";
21
22 /* elements of the GitHub API search */
23 /* Examples:
24 * Order by best match: &page=
25 * Order by stars desc.: +&sort=stars&order=desc&page=
26 * Based on language: +language:assembly&sort=stars&order=desc&page=
27 */
28 public final static String searchElements  = "+&sort=stars&order=desc&page=";
29
30 /* general GitHub repository search link */
31 public final static String gitRepo = "https://github.com/";
32
33 /* path for cloned repositories */
34 public static final String repoPath = "src/Repos/";
35
36 /* Timeout for beginning to clone the next repository in seconds */
37 public static final int cloneTimeOut = 300;
```

process.rb (preprocess analysis data from Ninka):

```
1
2  # retrieve all child directories (repositories) paths
3          Dir.entries('.').select{ |e| File.directory?(e) }.each do |folderpath|
4          folder_paths << folderpath
5  end
6
7  # check for all repositorie folders
8  folder_paths.each do |childfolder|
9          Dir.chdir(childfolder) do
10                 puts("currrently analyzing :" + childfolder)
11
12                 # retrieve all .license-file paths and store in array
13                 Dir['**/*'].select{|f| File.file?(f) }.each do |filepath|
14                         license_file_paths << filepath if filepath =~ /.*\.
                               license$/
15                 end
16
17                 # check for licenses in all existant paths
18                 license_file_paths.each do |licensefile|
19                 currentFile = File.foreach(licensefile)
20                         licenses.each do |currentLicense|
21                                 if currentFile.any?{ |l| l[currentLicense] }
22                                         license_appearences[i] = 1
23                                 end
24                                 i += 1
25                         end
26                         i = 0
27                 end
28
29                 # make license.txt for all found licenses
30                 license_appearences.each do |appearence|
31                         if appearence == 1
32                                 out_file0 = File.new(licenses[j] + ".txt", "w")
33                                 license_appearences[j] = false
34                         end
35                         j += 1
36                 end
37
38                 # clear for next iteration
39                 license_file_paths.clear
40                 license_appearences.clear
41                 i = 0
42                 j = 0
43
44         end
45 end
```

count.rb (count license occurrences after preprocessing):

```
1  #licenses
2  licenses = []
3  license_appearences = []
4
5  folder_paths = []
6  i = 0
7  j = 0
8  k = 0
9
10 # retrieve all child directories (repositories) paths
11 Dir.entries('.').select{ |e| File.directory?(e) }.each do |folderpath|
12         folder_paths << folderpath
13 end
14
15 # initialize license_appearences
16 licenses.each do |element|
17         license_appearences[j] = 0
18         j = j + 1
19 end
20
21 folder_paths.each do |path|
22         Dir.chdir(path) do
23                 Dir.entries('.').select{ |e| File.file?(e) }.each do |filepath|
24                         licenses.each do |currentLicense|
25                                 if filepath.include? currentLicense
26                                         if license_appearences[i] != nil
27                                                 license_appearences[i] =
                                                        license_appearences[i] + 1
28                                         end
29                                 end
30                                 i += 1
31                         end
32                         i = 0
33                 end
34         end
35 end
36
37 # output
38 licenses.each do |license|
39         puts(license, license_appearences[k])
40         puts()
41         k += 1
42 end
```

List of Figures

List of Tables

Bibliography

[1] Y. Wu *et al.*, "A method to detect license inconsistencies in large-scale open source projects." 12th Working Conference on Mining Software Repositories, Florence, Italy, 2015.

[2] C. Vendome *et al.*, "License usage and changes: A large-scale study of java projects on github." Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, 2015.

[3] R. Laemmel, "Software chrestomathies." Science of Computer Programming, 2013.

[4] "What is open source?." `http://opensource.com/resources/what-open-source` [abgerufen am 5.11.2015].

[5] B. Bruegge *et al.*, "Open source software: Eine oekonomische und technische analyse." Springer Verlag, Germany, 2004.

[6] B. f. Sicherheit i. d. Informationstechnik, "Fragen und antworten zu open source software." `https://www.bsi-fuer-buerger.de/BSIFB/DE/MeinPC/OpenSourceSoftware/FragenUndAntworten/fragenundantworten_node.html` [abgerufen am 5.11.2015].

[7] D. Wheeler, "The free-libre / open source software (floss) license slide," 2007.

[8] G. Inc., "Licenses." `http://choosealicense.com/licenses/` [abgerufen am 21.01.2016].

[9] "Working conference on mining software repositories." `http://msrconf.org/` [abgerufen am 1.10.2015].

[10] J.-M. Favre, R. Lämmel, *et al.*, "101companies: a community project on software technologies and software languages." Proceedings of TOOLS 2012, Springer, 2012.

[11] A. Engelfriet, "Choosing an open source license." Athens University of Economics and Business, 2010.

[12] J. Colazo *et al.*, "Impact of license choice on open source software development activity." Wiley InterScience, 2009.

[13] B. Balter, "Open source license usage on github.com," 2015. `https://github.com/blog/1964-license-usage-on-github-com` [abgerufen am 10.12.2015].

[14] B. Duck, "Top 20 open source licenses," 2014. `https://www.blackducksoftware.com/resources/data/top-20-open-source-licenses` [abgerufen am 10.12.2015].

[15] www.law.washington.edu, "Enforceability of open source licenses." `https://www.law.washington.edu/lta/swp/law/enforceability.html` [abgerufen am 06.01.2016].

[16] F. S. F. Software Freedom Conservancy, "The principles of community-oriented gpl enforcement," 2015. `https://sfconservancy.org/copyleft-compliance/principles.html` [abgerufen am 11.01.2016].

[17] K. M. Sandler, "Conservancy and fsf publish principles of copyleft enforcement," 2015. `https://sfconservancy.org/news/2015/oct/01/compliance-principles/` [abgerufen am 11.01.2016].

[18] KLBB, "Open source." `https://www.jbb.de/rechtsbegriffe/open-source` [abgerufen am 11.01.2016].

[19] A. Mathur *et al.*, "An empirical study of license violations in open source projects." Microsoft Research India, 2012.

[20] F. S. Foundation, "Gnu general public license," 2007. `http://www.gnu.org/licenses/gpl.html` [abgerufen am 23.03.2016].

[21] A. Mathur *et al.*, "An empirical study of license violations in open source projects." IEEE 35th Software Engineering Workshop, 2012.

[22] D. M. German *et al.*, "A sentence-matching method for automatic license identification of source code files." 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 2010.

[23] M. D. Penta *et al.*, "Identifying licensing of jar archives using a code-search approach." 12th Working Conference on Mining Software Repositories, Florence, Italy, 2015.

[24] P. Abate *et al.*, "Mining component repositories for installability issues." 12th Working Conference on Mining Software Repositories, Florence, Italy, 2015.

[25] I. Moura *et al.*, "Mining energy-aware commits." 12th Working Conference on Mining Software Repositories, Florence, Italy, 2015.

[26] G. Bavota *et al.*, "The market for open source: An intelligent virtual open source marketplace." Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, Antwerp, Belgium, 2014.

[27] L. foundation, "Spdx license list." v2.4. `https://spdx.org/licenses/` [abgerufen am 14.04.2016].

[28] N. McAllister, "Study: Most projects on github not open source licensed," 2013. `http://www.theregister.co.uk/2013/04/18/github_licensing_study/` [abgerufen am 23.03.2016].

[29] S. Vaughan-Nichols, "The fall of gpl and the rise of permissive open-source licenses." Linux and Open Source, 2014. `http://www.zdnet.com/article/the-fall-of-gpl-and-the-rise-of-permissive` [abgerufen am 23.03.2016].

[30] GNU, "Various licenses and comments about them." `http://www.gnu.org/licenses/license-list.en.html` [abgerufen am 05.04.2016].

[31] versioneye, "Verdict to gpl violation in germany," 2015. `https://blog.versioneye.com/2015/09/21/`

judgment-to-gpl-violation-in-germany/ [abgerufen am 20.04.2016].

[32] A. Stiller, "The open source trials: hanging in the legal balance of copyright and copyleft," 2011. http://www.visionmobile.com/blog/2011/03/the-open-source-trials-hanging-in-the-legal-balance-of-copyright-and-co [abgerufen am 20.04.2016].

[33] P. A. QUESTIONS, "Gpl compatibility." http://www.apache.org/legal/resolved.html [abgerufen am 05.04.2016].

[34] Wiki, "Comparison of free and open-source software licenses." https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses [abgerufen am 05.04.2016].

[35] Sun-Times, "Microsoft ceo takes launch break with the sun-times," 2001. http://web.archive.org/web/20010615205548/http://suntimes.com/output/tech/cst-fin-micro01.html [abgerufen am 21.04.2016].

[36] H. Baldwin, "4 reasons companies say yes to open source," 2014. http://www.computerworld.com/article/2486991/app-development-4-reasons-companies-say-yes-to-open-source.html [abgerufen am 03.03.2016].

[37] K. Noyes, "10 reasons open source is good for business," 2010. http://www.pcworld.com/article/209891/10_reasons_open_source_is_good_for_business.html [abgerufen am 03.03.2016].

[38] Z. Shawn, "Why i (a/l)gpl." http://zedshaw.com/archive/why-i-algpl/ [abgerufen am 03.03.2016].

[39] Z. Shawn, "Why i (a/l)gpl." http://zedshaw.com/archive/is-bsd-the-new-gpl/ [abgerufen am 03.03.2016].

[40] R. Gobeille, "The fossology project." International Working Conference on Mining Software Repositories, MSR 2008, Leipzig, Germany, 2008.

[41] T. Tuunanen *et al.*, "Automated software license analysis. autom. softw. eng.," 2009.

[42] C. Boldyreff *et al.*, "Open source ecosystems: Diverse communities interacting." 5th IFIP WG 2.13 International Conference on Open Source Systems, Sweden, 2009.