JYU DISSERTATIONS 371

Timo Tuunanen

Tool Support for Open Source Software License Compliance

The First Two Decades of the Millennium

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 371

Timo Tuunanen

Tool Support for Open Source Software License Compliance

The First Two Decades of the Millennium

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi kesäkuun 4. päivänä 2021 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
on June 4, 2021 at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2021

Editors
Timo Männikkö
Faculty of Information Technology, University of Jyväskylä
Ville Korkiakangas
Open Science Centre, University of Jyväskylä

Permanent link to this publication: http://urn.fi/URN:ISBN:978-951-39-8596-7

ISBN 978-951-39-8596-7 (PDF)
URN:ISBN:978-951-39-8596-7
ISSN 2489-9003

ABSTRACT

Tuunanen, Timo
Tool Support for Open Source Software License Compliance: The First Two Decades of the Millennium
Jyväskylä: University of Jyväskylä, 2021, 116 p.
(JYU Dissertations
ISSN 2489-9003; 371)
ISBN 978-951-39-8596-7 (PDF)

Open source software reuse enables developers to leverage past accomplishments while facilitating improvements in software productivity and quality. However, licenses of the reused software need to be considered to be compliant with the license terms, thus mitigating intellectual property right risks related to such reuse. Identifying under which license(s) an open source software is provided and understanding the terms of these licenses is not trivial, especially when dealing with substantial reuse, which is common in modern software development. As reused software is often large, automated license analysis is needed to address these issues and to support users in the license compliant reuse of open source software.

This study aims to provide a comprehensive view on the automated features and methods that assist in open source license compliance. It describes the automated tools and methods of license compliance, here spanning two decades of research. The empirical study consists of two cycles: In the design cycle, we identified the critical user needs for automated license compliance, such as the license identification of source files and license compatibility analysis, and created a novel approach ASLA (Automated Software License Analyzer) that supports these needs. In the review cycle, which consisted of a systematic literature review, we describe how automated license compliance software has evolved since the introduction of ASLA. We identified new user needs from the included literature, such as an identification of the origin of the OSS and needs related to comprehension of OSS licenses. Also, we list the features that were introduced after the design cycle.

As a conclusion, there is a clear need for automated OSS license compliance tools since the amount and reuse of OSS has increased significantly over the past 10 years. Based on the information of these two cycles, we merged and listed a set of user needs, which are composed of 16 individual needs. It became evident that no tool is available that would support all of these needs. Whereas license identification and compatibility analysis are fields that have the most mature solutions in the license compliance process, future research is needed to improve features related to copyright extraction and the integration of existing features as part of development process.

Keywords: Open source software, License compliance, Compliance analysis, Tool support

TIIVISTELMÄ (ABSTRACT IN FINNISH)

Tuunanen, Timo
Työkalutuki avoimen lähdekoodin lisenssien noudattamiseksi: Vuosituhannen kaksi ensimmäistä vuosikymmentä
Jyväskylä: University of Jyväskylä, 2021, 116 s.
(JYU Dissertations
ISSN 2489-9003; 371)
ISBN 978-951-39-8596-7 (PDF)

Avoimen lähdekoodin uudelleenkäyttö mahdollistaa ohjelmistokehityksen tuottavuuden ja ohjelmistojen laadun parantamisen. Uudelleenkäytön yhteydessä ohjelmistojen lisenssien asettamia ehtoja tulee kuitenkin noudattaa, jotta immateriaalioikeuksiin liittyvät riskit voidaan minimoida. Erityisesti modernissa, runsaaseen uudelleenkäyttöön pohjautuvassa ohjelmistokehityksessä lisenssien tunnistaminen ja ymmärtäminen on haastavaa, koska uudelleenkäytettäviä ohjelmistoja on paljon ja ne ovat usein laajoja. Lisenssien noudattamista ja tehokasta uudellenkäyttöä tukemaan tarvitaan automatisoituja menetelmiä.

Tämä tutkielma pyrkii esittämään kattavan kuvan automatisoiduista toiminnallisuuksista ja metodeista, jotka tukevat lisenssien noudattamista. Se esittelee automatisoituja työkaluja ja menetelmiä kahden vuosikymmenen ajalta. Empiirinen osuus koostuu kahdesta syklistä: suunnittelusyklistä ja katsaussyklistä. Suunnittelusyklissä tunnistettiin automatisoituun lisenssien noudattamiseen liittyvät kriittiset käyttäjätarpeet, kuten esimerkiksi lisenssien tunnistaminen lähdekooditiedostoista ja lisenssien yhteensopivuusanalyysi sekä kehitettiin tarpeita tukeva uusi ohjelmisto ASLA (Automated Software License Analyzer). Katsaussykli muodostuu systemaattisesta kirjallisuuskatsauksesta, jossa kuvataan kuinka työkalut ja menetelmät ovat kehittyneet ASLA:n julkaisemisen jälkeen. Siinä tunnistettiin sisällytetystä kirjallisuudesta käyttäjätarpeita, joita ei ollut tunnistettu ensimmäisessä syklissä. Näitä ovat esimerkiksi avoimen lähdekoodin ohjelmiston alkuperän tunnistaminen ja lisenssiehtojen ymmärtämisen tukeminen. Lisäksi siinä listataan toiminnallisuuksia, jotka on esitelty sunnittelusyklin jälkeen.

Yhteenvetona voidaan todeta, että automatisoidulle lisenssianalyysille on selkeä tarve, koska sekä avoimen lähdekoodin määrä että sen uudelleenkäyttö on kasvanut huomattavasti viimeisen 10 vuoden aikana. Kahden tutkimyssyklin pohjalta yhdistettiin ja listattiin yhteensä 16 itsenäistä käyttäjätarvetta. Tutkimuksessa kävi selväksi, että mikään yksittäinen työkalu ei tue kaikkia näitä tarpeita. Lisenssien tunnistamiseen ja yhteensopivuusanalyysiin liittyvät toiminnallisuudet ovat parhaiten tuettuja. Jatkotutkimusta tarvitaan erityisesti parantamaan toiminnallisuuksia, jotka liittyvät tekijänoikeustietojen keräämiseen lähdekoodista ja olemassa olevien toiminnallisuuksien liittämiseksi osaksi kehitysprosessia.

Avainsanat: Avoin lähdekoodi, Lisenssien noudattaminen, Noudattamisanalyysi, Työkalutuki

Author

Timo Tuunanen
Faculty of Information Technology
University of Jyväskylä
Finland

Supervisors

Professor Tommi Kärkkäinen
Faculty of Information Technology
University of Jyväskylä
Finland

Professor Tommi Mikkonen
Department of Computer Science
University of Helsinki
Finland

Reviewers

Professor Daniel M. German
Department of Computer Science
University of Victoria
Canada

Associate Professor Georgia M. Kapitsaki
Department of Computer Science
University of Cyprus
Cyprus

Opponent

Professor Kari Smolander
Software Engineering
LUT University
Finland

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Professor Tommi Kärkkäinen, Professor Tommi Mikkonen, and Doctor Jussi Koskinen, for their encouragement, discussions, and feedback over the years.

I also would like to thank Professor Daniel M. German and Associate Professor Georgia M. Kapitsaki for their valuable feedback as reviewers of this dissertation.

The initial idea for the automated license analysis and many of the user needs that were implemented in ASLA came from mr. Dietmar Tallroth in the early years of the millennium. I would like to give my gratitude to him, as this dissertation would not exist without his involvement.

I hereby also thank all my family and friends for their encouragement, support, and understanding.

Figures 4, 5, 8, 9, and 10 are material from: Tuunanen, T., Koskinen, J. & Kärkkäinen, T., Automated software license analysis, Automated Software Engineering 16, published 2009, Springer Nature Switzerland AG.

Espoo 8th of March 2021. Timo Tuunanen

NOMENCLATURE

ACM Association for Computing Machinery.

AGPL Affero General Public License.

ASLA Automated Software License Analyzer.

BSD Family of permissive free software licenses. The original BSD license was used for its namesake, the Berkeley Software Distribution (BSD), a Unix-like operating system.

CBDG Concrete Build Dependency Graph.

CDDL Common Development and Distribution License.

CONTU US Commission on New Technological Uses of Copyrighted Works.

COTS Commercial off-the-shelf software component.

DFSG Debian Free Software Guidelines.

DRAT Distributed Release Audit Tool.

DSR Design Science Research.

EPL Eclipse Public License.

FAQ Frequently Asked Questions.

FLC Formalized License Compliance.

FLOSS Free/Libre and Open Source Software.

FOSS Free and Open Source Software.

FS Free Software.

GCC GNU Compiler Collection.

GPL GNU General Public License. Different versions of GPL licenses are referred with following notation: (L)GPLvX(+), for example GPLv2 meaning GPL version 2 or GPLv2+ meaning GPL version 2 or any later version.

HTML Hypertext Markup Language.

IEEE Institute of Electrical and Electronics Engineers.

IPR Intellectual Property Rights.

ISO International Organization for Standardization

LGPL GNU Lesser General Public License.

MIT Permissive free software license originating at the Massachusetts Institute of Technology (MIT).

MPL Mozilla Public License.

OBIE Ontology-Based Information Extraction.

ODRL Open Digital Rights Language.

OSI Open Source Initiative.

OSLC Open Source License Checker.

OSS Open Source Software.

RAT Release Audit Tool.

RDF Resource Description Framework.

RQ Research Question.

SLR Systematic Literature Review.

SPDX Software Package Data Exchange.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Software systems have become increasingly complicated and large. For instance, Mozilla Firefox browser, which licenses we analyzed in 2009, had 2 million lines of code at the time of analysis (Tuunanen et al. 2009). In 2020, it is consisted of about 20 million lines of code (Abadie and Ledru 2020). Since it would be impractical to build large and complex systems from scratch, most systems are constructed using preexisting components (Hartmann et al. 2008). Software reuse can increase productivity by reducing development costs and minimizing schedule overruns, because fewer lines of code need to be written (Kim and Stohr 1998). Modern software development rely heavily on reuse, and often, only a small portion of code is written by the developers, with the rest being reused (Mikkonen and Taivalsaari 2019).

Already in early 1980s, it was identified that the majority of the produced source code appeared to be common, generic code that could be reused between different applications (Jones 1984; Lanergan and Grasso 1984). Nowadays, reusable code comes in all shapes and sizes, for example, in code snippets, software libraries, individual applications, or even as complete operating systems. There are two main sources of reusable code: proprietary software, often in the form of commercial off-the-shelf (COTS) components, and open source software (OSS). The emergence of the software-as-a-service model, Internet-based developer forums, and OSS repositories have enabled an approach in which people routinely use ready-made solutions from online sources for all kinds of problems (Mikkonen and Taivalsaari 2019). This has significantly increased the importance of reusable OSS over proprietary software, and as a result, almost all companies use open source components in their software (Franch et al. 2013).

The amount of OSS code has increased tremendously over the last decade. It has been estimated that in 2009, there were in total about 5 billion lines of OSS code, whereas at the end of 2018, the estimation had increased to 17 billion lines of code (Dorner et al. 2020). The statistics by GitHub (2020)[1], the popular open source hosting site, reveal the current significance and magnitude of OSS development: during one year, almost 2 billion contributions (25% increase over

[1] https://github.com/ accessed Jan 18, 2021

the previous year) were added to GitHub, which has over 56 million registered developers. During the same time span, October 2019 – September 2020, more than 60 million new repositories (35% increase over previous year) were created on GitHub (2020). Also, the number of libraries designed especially for reuse has enlarged, and developers often rely on these libraries to provide a specific functionality in their applications (Kula et al. 2018). For instance, in 2010, Maven Central[2] contained over 260,000 reusable libraries, whereas in 2016, this collection of libraries had risen to almost 1.7 million, and at the end of 2020, it had increased to over 6 million.

During the early years of open source, the contributors of OSS projects were mostly individuals and communities (Fitzgerald 2006). However, nowadays, companies have become active contributors and even companies such as Microsoft that used to oppose open source have changed their opinion. In his famous statement in 2001, Microsoft CEO Steve Ballmer said that the most notable open source project at the time, Linux, "is a cancer that attaches itself in an intellectual property sense to everything it touches" (Greene 2001). However, some 15 years later, Microsoft had grown to GitHub's largest open source contributor (Weissman 2016). In 2020, Microsoft president Brad Smith admitted that "Microsoft was on the wrong side of history when open source exploded at the beginning of the century" (Hanson 2020). Other top open source contributors include companies such as Google, Red Hat, IBM, and Intel (Asay 2018).

Companies increasingly realize the benefits of using OSS components in their products; these benefits include short time-to-market, reduced development and maintenance costs, and customization capabilities (Franch et al. 2013). One recent example of the efficiency of open source reuse is the application called Koronavilkku[3] (version 1.3.0.), which is used in Finland to detect and contact people that may have been exposed to the coronavirus (COVID-19). It was developed and tested by Solita Inc., and it was reviewed and accepted by Finnish authorities over a time span of six months (Finnish Institute for Health and Welfare 2020). The Android version of Koronavilkku employs over 100 open source components. A list of these components and their respective license texts are found under the application's settings menu.

Since software code is protected by intellectual property rights (copyright and in some cases with patents and trademarks), it must be reused in compliance with its license. Licenses provide authorization to use or exploit copyright protected work. Open source licenses allow, for instance, modifications to the source and creation of derived works. Unfortunately, there are a lot of misunderstandings about the characteristics of OSS licensing. This problem is amplified by a large number of existing open source licenses as there are more than 100 Open Source Initiative (OSI) -approved licenses available (Open Source Initiative 2020a). This leads to a situation where developers may not fully understand the terms of open source licenses and the differences between these licenses (Almeida et al. 2019). Being compliant with licenses rights and obligations, especially when

[2] https://mvnrepository.com/repos/central accessed Jan 18, 2021

[3] https://koronavilkku.fi accessed Jan 18, 2021

combining software under different licenses, may be challenging. Also, as some open source licenses are incompatible, combining components with different licenses may not even be possible. To fully understand open source licensing and its related challenges, the background information about the underlying legal framework and intellectual property rights must be understood.

Reusers need to control their reuse of OSS to avoid common threats, such as license non-compliance leading to copyright and patent infringement, which can result in legal consequences or product recalls (Ruffin and Ebert 2004; Synopsis 2018). License compliance can be addressed using a three-step process: identify the potentially reused software and its licenses, approve their usage, and satisfy the license obligations (Haddad 2019). However, identifying the license of a piece of software is often non-trivial since (i) licenses may not be explicitly stated (Kapitsaki et al. 2015) or (ii) there may be inconsistencies between the stated license and that of individual source code files (Vendome et al. 2018). Thus, reliable and detailed information about the license(s) of the software component must be gathered from the source code. The approval stage ensures that the license terms are understood and that no OSS is reused where licensing terms are violated or non-approved licenses are used. Satisfying the license obligations typically requires acknowledgement documentation, reproduction of license text, and, in case of copyleft licenses, making the copies of the source code of the OSS component available.

This study aims to provide a comprehensive view of user needs and automated features and the methods that assist a software reuser in license compliance. User needs refer to the requirements that add value to the user (i.e., the reuser) of a license compliance tool, features refer to the features of these tools, and methods refer to, for example, the processes how the tools are used or other models and practices that assist in compliance assurance. The comprehensive picture that we aim to give includes the legal background of software licenses, introduction of open source licenses, license compliance process, and challenges of constructing automated tools that assist in compliance, along with looking at the status of existing tools' features and implementation approaches.

1.1 Methodology and Research Questions

The overall research process of this thesis was slightly unorthodox as the study was performed in two separate cycles. The first cycle was conducted between 2004 and 2009 and the second cycle in 2020.

Cycle 1 (see Figure 1), the design cycle, covered the first decade (2000 – 2009, numbering like, e.g., in c-language) of the millennium and consisted of creation of a reverse engineering approach and its implementation, called ASLA (Automated Software License Analyzer). These were constructed to fulfill the need for automated license compliance assurance. The results were published in three separate papers (Tuunanen et al. 2006a,b, 2009). In this cycle, a design

science methodology (Hevner 2007; Peffers et al. 2007) was followed, consisting of four parts: (1) understanding the environment, (2) building artifacts, (3) evaluating artifacts, and (4) communicating outcomes in publications. As a result of understanding the environment, the user needs for the license analyzer and the implementation challenges of fulfilling these needs were discovered. The implementation and enhancements of ASLA and its evaluation in terms of features and its performance fulfill the the artifact building and evaluation parts of design science methodology. The three publications describing these steps fulfill the communicating of the outcomes. The results of the design cycle are presented in Chapter 4.

Cycle 2 (see Figure 1), the review cycle, covered the second decade (2010 onwards) of the millennium and was based on the hypothesis that the number of automated tools that assist in license compliance assurance has been increasing as the amount of OSS and its reuse have also been increased substantially during the past decade. As there is no up-to-date comprehensive picture of the state of these tools, it is important to describe and organise this information. Therefore, in this research cycle, we focus on what automated tools are available in 2020, how user needs have evolved, and what features have been improved and discovered since the introduction of ASLA. The review cycle is done as a systematic literature review (SLR) (Kitchenham et al. 2015) by: (1) identifying the relevant research, (2) selecting studies, (3) assessing study quality, (4) extracting data from selected studies, and (5) synthesizing the results. The results of the review cycle are presented in Chapter 5.

We also do a reflective comparison, where the evolution of license compliance tools, user needs, and features are analyzed in comparison to ASLA's approach. This reflective comparison is presented in Chapter 6.

As we want to have a comprehensive picture of the state of automated open source license compliance, it is important to know what user needs have been discovered. This leads to first research question, which is stated as follows:

RQ1 What are the user needs to fulfill automated open source license compliance?

As our research consists of design and review cycles, it leads to two sub-questions:

RQ1.1 How does the design cycle contribute to this question?
RQ1.2 How does the review cycle contribute to this question?

Also, as we are interested in the actual features or methods for license compliance, we state our next research question as follows:

RQ2 What software features are needed to fulfill the user needs?

Also, for this question, we need to state two sub-questions:

RQ2.1 How does the design cycle contribute to this question?
RQ2.2 How does the review cycle contribute to this question?

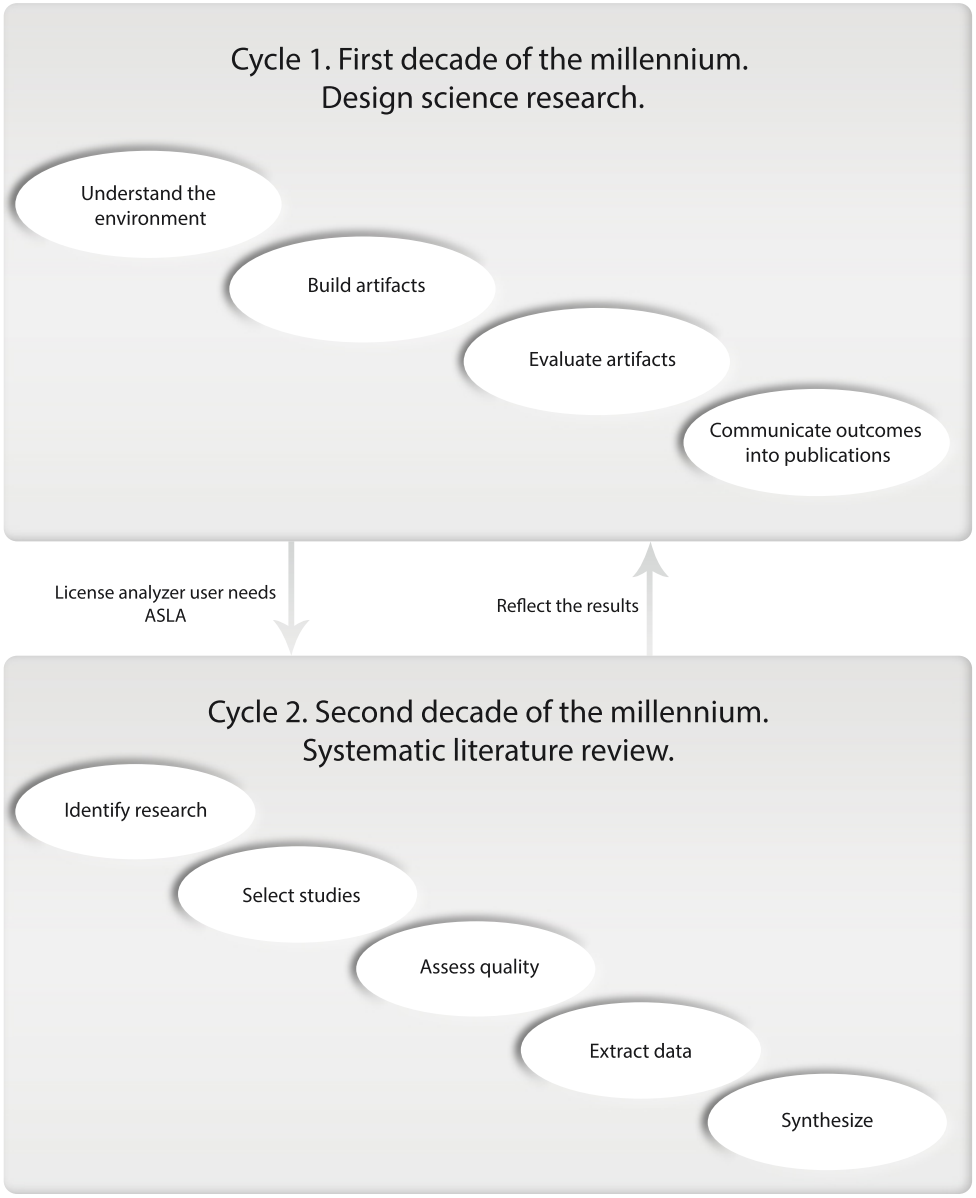

FIGURE 1 Two cycles of this study.

1.2 Overview of Results

The main research questions stated above were answered by conducting and summarizing the findings from the two cycles of this study. The results present a comprehensive picture of the status of automated tool support of OSS license compliance in terms of the identified user needs and available features.

During the design cycle, we identified 12 user needs for automated license compliance (see Table 2). These include, for example, the license identification of each source file and the automated license compatibility analysis. Our implementation, ASLA, supports all of these needs. The evaluation of ASLA revealed that it enables the analysis of large OSS programs in a reasonable time and provides valuable information for meeting licence compliance.

After the search and selection process, the review cycle produced 53 relevant publications. What was slightly surprising was the fact that the yearly amount of research related to automated OSS license compliance has not increased during the past 10 years. It could have been expected that as the amount of OSS and relevance of OSS reuse has increased dramatically, the research would have followed this trend.

We collected seven new user needs from the relevant publications in the review cycle that were not identified in the design cycle. Even though the design cycle identified most of the user needs for automated license compliance, the review cycle revealed fields that were not present in the design cycle. These include the identification of the origin of the software and automated license comprehension. We also collected and organized the existing features for automated license compliance that were identified during the review cycle. The features described using scientifically sound methods (from 20 papers) are summarized in Table 10. Notable advancements in the features found in the review cycle include, for example, a new license identification technique in the tool called Ninka (German et al. 2010b), various code clone detection features used for identification of the origin of the software, and features related to the creation, validation, and examination of Software Package Data Exchange (SPDX) files. SPDX is a standard format used for communicating the components, licenses, and copyrights associated with a software package (SPDX Workgroup 2020a).

Based on the design and review cycles, we merged, organized, and listed a total of 16 user needs for an automated license compliance tool; these are presented in Table 12. No single tool currently implements features that cover all of these needs. Especially, the research related to satisfying the license obligations seems to be in its early stages.

1.3 Structure of the dissertation

Chapter 2 introduces intellectual property rights, a legal instrument that protects the rights of the authors of software, and open source licenses, which is the mechanism that allows the exploitation of these rights. The details of open source license compliance and the related process to address the legal issues that developers face when reusing OSS, is described in Chapter 3. The results of the study are presented in two chapters as the study consists of two research cycles. The results from the design cycle and the DSR approach we used for the cycle are presented in Chapter 4. Chapter 5 describes the results of the review cycle, along with a detailed description of the research protocol used in the SLR. A reflective synthesis of the results from the two cycles are presented in Chapter 6. In Chapter 7, we provide the conclusions of the study.

2 INTELLECTUAL PROPERTY RIGHTS AND OSS LICENSES

In this chapter, we will describe intellectual property rights from a software licensing point of view. This includes the basics of copyrights, patents, and trademarks. Also, we describe the characteristics of open source licenses and how they allow the exploitation of intellectual property rights.

2.1 Intellectual property rights

Creations of the mind, such as inventions; literary and artistic works; designs; and symbols, names, and images used in commerce are considered intellectual property (WIPO 2020b). The importance of intellectual property was first recognized in the Paris Convention for the Protection of Industrial Property (1883) and the Berne Convention for the Protection of Literary and Artistic Works (1886) (the Berne Convention) (WIPO 2016).

2.1.1 Overview

Intellectual Property Rights (IPR) allow the creators or owners of copyrighted works, patents, and trademarks to benefit from their work, thus forming the foundation of the software industry. Technical ideas, forms, individual expressions, and other kinds of immaterial values are covered by the ownership of IPR (Hellstadius 2010). These rights are defined in Article 27 of the Universal Declaration of Human Rights, which "provides for the right to benefit from the protection of moral and material interests resulting from authorship of scientific, literary, or artistic productions" (United Nations 2020).

Intellectual property is different from other property in at least three ways (Weckert 1997): First, owning an idea or other abstract object such as rights to software is not similar to owning a physical object. Someone can take someone else's software by copying it, and even so, both the owner and user can use and

enjoy it. Therefore, intellectual property is non-exclusive. Second, to what extent is an idea unique? They come from anywhere, and most likely, any idea that one has is not that person's alone. For example, most of the ideas presented in this thesis come from someone other than us. At best, when one is "original," he or she expresses an idea in a novel way. The third difference can be drawn between moral rights and economic rights to intellectual property. A moral right is the right to acknowledgement as the author or creator. A economic right is the right to profit financially from the property. As intellectual properties are non-exclusive, the main issue when infringing on owners intellectual property rights is not taking something from the owner or creator and thereby depriving him or her of access to it (Weckert 1997). The issue is the copying of the expression or the idea.

There is a fundamental difference in copyright law between an idea and an expression (Galler 1995). A copyright protects only expressions of an idea. Ideas such as mathematical formulae themself do not fall under copyright protection (Weckert 1997). Whereas the copyright protects the expressions, a patent is an exclusive right granted for an invention and to commercially benefit from that invention (Hellstadius 2010). Of the IPRs copyrights and patents are the most meaningful in case of computer software (Menell 1989). Also, in the context of OSS, trademarks must be considered to some extent since some OSS licenses take into account the use of trademarks. A trademark is a sign that distinguishes the goods or services and trademark law prevents competitors from using similar marks to identify their own goods and services (Georgievski 2020). In summary, IPRs do not grant the exclusive use of owned property. They are concerned more with controlling who can use the property and who gets acknowledgement and financial reward from that use (Weckert 1997).

2.1.2 Copyrights

A copyright is a right given to authors of original[1] works, such as poems, musical compositions, movies, and computer programs. A copyright comes into existence immediately upon the production of the work in a "tangible medium of expression" and lasts for the life of the author plus for a minimum period of 50 years after the death of the author (WIPO 2020b; WTO 2020). Copyright protection is obtained without any formalities in countries that have agreed to the Berne Convention (WIPO 2020a), meaning that the registration or deposit of copies is not required. For example, copyright for computer software is created immediately as the software code is written.

It was debated in the 1970s and 1980s whether the patent system, the copyright system, or a *sui generis*[2] system should provide protection for computer

[1] "The threshold of originality is a concept in copyright law that is used to assess whether a particular work can be copyrighted. It is used to distinguish works that are sufficiently original to warrant copyright protection from those that are not (Wikipedia 2020)."

[2] *Sui generis* is a Latin phrase that means "of its/his/her/their own kind, in a class by itself," therefore "unique."

software (WIPO 2020a). In 1974, US Commission on New Technological Uses of Copyrighted Works (CONTU) (1978) decided that "software is a proper subject matter of copyright to the extent that it embodies its authors' original creation" (Keplinger 1981). CONTU's decision and other discussions resulted in the generally accepted principle that "computer programs should be protected by copyright, whereas apparatus using computer software or software-related inventions should be protected by patents" (WIPO 2020a). In software programs, copyright protection extends to the overall structure, sequence, and organization of an application program and also to the overall structure sequence and arrangement of the screens, text, and artwork (i.e., the audiovisual display in general) (Menell 1989). Most countries accept copyright protection of computer software and international treaties guarantee its consistent application worldwide (WIPO 2020a).

The author of the copyrighted work is allowed to exclusively explore all the usages of its creation, excluding others from using it without proper authorization (Pina 2011). Two types of rights emerge: they allow the rights owner to derive financial reward from the use of their works by others (economic rights), and protect the non-economic interests of the author (moral rights) (Weckert 1997).

While the cost of creating a work subject to copyright protection, such as a book, movie, or a computer program, is often high, the cost of reproducing, whether by the creator or by those to whom he or she has made it available is often low (Landes and Posner 1989). This is the core of the copyright protection: the right of the copyright's owner to prevent others from making copies and giving these copies to others, which is also referred to as the right of reproduction and distribution (Lemley 1994). Also, copyrights provide technical procedures for the enforcement of these rights (O'Hare 1982). From an economic perspective, an owner of intellectual property is deemed to lose if the property is copied or used without authorization (Weckert 1997). Unauthorized copying, that is, software piracy has traditionally been very common. In 2000, essentially all new software was pirated in some countries, whereas in some other countries, less than 40% of software was pirated (Marron and Steel 2000). The copyright owner also has economic rights related to right of performance, right of public transmission, right of presentation, and right of lending. However, these are less relevant in the case of computer programs.

Rigamonti (2006) describes the standard set of moral rights as follows: "author's right to claim authorship (right of attribution), the right to object to modifications of the work (right of integrity), the right to decide when and how the work in question will be published (right of disclosure), and the right to withdraw a work after publication (right of withdrawal)." Justification for the existence of moral rights is that the creator's labour was put into the original work or that the creator just deserves some reward for having an idea and developing it (Weckert 1997). From a moral perspective, computer programs have a special characteristic as the adaptation and improvement of software can only be developed if the source code is available (Pina 2011).

As a summary of economic and moral rights for computer programs, the owner of the copyright can authorize or prevent the creation of copies of the soft-

ware, usage of the software, creation of combined, modified or derivative works of the software, and distribution of the software (Freibrun 1995). These rights form the fundamental elements of OSS licenses described in more detail in Subsection 2.2.

2.1.3 Patents

A patent is an intellectual property right for inventions, that is, in the devices or processes that perform a useful industrial applicable function. A patent effectively grants the inventor a limited monopoly, that is, a right to prevent others from producing, using, or selling the invention (Hall and MacGarvie 2010). A patent is generally granted after completing an examination procedure by a government agency, and the protection is granted for a limited period, generally 20 years from the filing date of the application (WIPO 2020b). There are no internationally harmonized laws related to software patents (Hellstadius 2010). Some countries allow software patents as such, while others have adopted approaches that recognize inventions assisted by software (WIPO 2020a).

To be granted a patent, there must be an invention that surpasses a threshold test. Various countries judge the invention threshold differently (Hellstadius 2010). The criteria of novelty, inventive step/non-obviousness, and industrial applicability/utility must be fulfilled. These requirements are nearly universally recognized, but their application may differ between countries (Hellstadius 2010). Exclusions are done for matters that are not patent eligible, for example, abstract ideas or mathematical formulae. Whereas copyright protection is (almost) global, patents must by applied separately for different countries or areas, such as the European Union.

In the case of software, patents protect the technical purpose of the program, which is more about the technical output than the actual code itself (Hellstadius 2010). The main difference between copyright and patents in information technology is that the copyright protects original computer programs against unauthorized copying, whereas patents cover the underlying ideas, procedures, and methods of operation. Software patents have been granted in the U.S. since the 1970s (Evans and Layne-Farrar 2004). On the other hand, even though the European Patent Office (2000) specifically declared that software is not patentable in the European Union, software patents have been granted as computer implemented inventions (Hall and MacGarvie 2010).

2.1.4 Trademarks

A trademark refers to a word, phrase, symbol, or design that is used to identify the producer of goods or services sold, and to distinguish them from the goods or services of others (Georgievski 2020). By registering trademarks, the owners of such trademarks secure legal protection of their investment in marketing, reputation for quality, brand names, and distribution channels (Fosfuri et al. 2008).

For example, even though Linux is a clone of the operating system UNIX, as

stated in the official README -file of the Linux kernel, it cannot claim to be UNIX since UNIX is a registered trademark of the Open Group. On the other hand, for example, Mac OS X (based on BSD, another UNIX -like operating system) is certified UNIX, so it is UNIX both in name and in functionality. Trademark rights can last indefinitely (unlike copyrights and patents), if the trademark owner continues to use the mark and renews its registration (Georgievski 2020).

2.1.5 Licensing IPR protected work

The rightful owner (i.e., copyright, patent, or trademark holder) of a work can provide authorization for others to use or exploit that work. Such authorizations are commonly referred to as licenses (Lindman et al. 2011). The terms under which a software system (either its code or binary) can legally be distributed, modified, and reused are defined by software licenses (Vendome et al. 2018). Traditionally, proprietary software is distributed in binary form, granting users limited rights to use the software (Von Krogh and Von Hippel 2003). Typically, only the authors who created the software can legally edit, inspect, change and enhance it. On the other hand, open source licensed software is computer software where the source code is made available by the author(s) to run, copy, distribute, study, change, and improve (Free Software Foundation 2019). Open source licenses allowing such actions are described in more detail below.

2.2 Open source software licenses

Open source refers to a model of software development where the source code is made publicly available and free of charge. Also, interested parties have the right to modify and extend these programs. This is made possible by open source licenses, which grant rights otherwise protected by international IPR laws.

2.2.1 History and overview

Many of the concepts that characterize the open source movement have been around since the beginning of the computing era (Aksulu and Wade 2010). These include concepts such as peer production, shared code, and software as a public good. However, the use of the term “open source” was not introduced until around the turn of the millennium.

A general trend in 1980s in the software world was toward the development of proprietary software packages and the release of software protected under licenses that prevented it from being studied or modified by others (Von Krogh and Von Hippel 2003). In 1983, Richard Stallman launched the GNU Project to write a complete operating system free from the constraints of its source code usage (Free Software Foundation 2016). Stallman also published the GNU Manifesto (Stallman 1985) to outline the GNU project’s purpose and explain the importance of

free software: preserve free access for the software and its code (Von Krogh and Von Hippel 2003). From mid 1980s to late 1990s, the term "free software" (FS) was mainly used. The term open source software was introduced in 1998 (Peterson 2018). Licensing practices of OSS includes essentially the same as those pioneered by the FS movement. OSS differs from FS movement primarily on philosophical grounds, preferring to emphasize the practical benefits of such licensing practices over issues regarding the moral rightness and importance of granting users the freedoms (Von Krogh and Von Hippel 2003). According to Peterson (2018), "a term was needed that focuses on the key issue of source code and that does not immediately confuse those new to the concept." Nowadays, also the terms "free and open source software" (FOSS) (Ebert 2008) and "free/libre and open source software" (FLOSS) (Harutyunyan et al. 2019) are being used. However, we use the term "open source" in this dissertation for its simplicity and because it is commonly used by scholars to refer to FS and OSS (Von Krogh and Von Hippel 2003).

In addition to the actual source code, in the core of the OSS is the OSS licenses as they dictate how open source can be used from a legal standpoint. Open source licenses have very specific characteristics as defined in Open Source Definition (Open Source Initiative 2020d). For instance, "the license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale. The program must include source code, and must allow distribution in source code as well as compiled form. The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties." With these terms, copyright owners allow the exploitation of most of their economic and moral rights. One exception is the right to claim authorship: OSS typically has an author or authors who own the copyright.

Conditions related to freedom to use, modify, and redistribute the original version of the OSS, are essentially the same in all open source licenses (Sen et al. 2011). Some open source licenses (e.g., Apache 2.0, GPLv3) include patent clauses, which grant recipients a license to any patents covering the given software. Other open source licenses (e.g., BSD, MIT, GPLv2) have no mention of patents whatsoever. Meeker (2017) mentions that "for these licenses, courts may use the doctrine of 'implied license' to find that recipients are still licensed and protected from any patent infringement allegation arising from using the licensed software product. By doing this, courts prevent licensors for suing for patent infringement for using the very software they have licensed." However, it is difficult to estimate the real risk of patent infringement, because many commonly used OSS components already infringe patents (Välimäki and Oksanen 2005).

Small minority of open source licenses mention trademarks, because use of a trademark requires monitoring and quality control by the trademark owner. Therefore, there is nothing "open" about trademark use. When the trademarks

TABLE 1 Popular and widely used open source licenses by OSI.

License	Permissive	Strong copyleft	Weak copyleft
Apache License 2.0 (Apache 2.0)	x	-	-
3-clause BSD license (BSD-3-Clause)	x	-	-
2-clause BSD license (BSD-2-Clause)	x	-	-
GNU General Public License, version 2 (GPL-2.0)	-	x	-
GNU General Public License, version 3 (GPL-3.0)	-	x	-
GNU Lesser General Public License, version 2 (LGPL-2.0)	-	-	x
GNU Lesser General Public License, version 2.1 (LGPL-2.1)	-	-	x
GNU Lesser General Public License, version 3 (LGPL-3.0)	-	-	x
MIT license (MIT)	x	-	-
Mozilla Public License 2.0 (MPL-2.0)	-	-	x
Common Development and Distribution License 1.0 (CDDL-1.0)	-	-	x
Eclipse Public License 2.0 (EPL-2.0)	-	-	x

are addressed specifically (e.g. Apache 2.0), the reference to trademarks is not to grant trademark rights. Trademarks are mentioned to expressly clarify that no rights to use the names or trademarks are granted (Hashimoto and Portner 2020).

Various OSS licenses have one key characteristic that differentiates the licenses. This is the degree of restrictions related to the ability to redistribute modified version(s) of the OSS or derivative work(s) based on the OSS (Fershtman and Gandal 2007). Lerner and Tirole (2005) propose three classes of OSS licenses based on the restrictiveness of redistribution rights: unrestrictive, restrictive, and highly restrictive. Fershtman and Gandal (2007) distinguish between three levels of (relative) license restrictiveness: non-restrictive, moderately restrictive, and very restrictive, whereas Phipps (2013) proposes the terms nonreciprocal, file-scoped reciprocal, and project-scoped reciprocal. To use the terminology familiar to OSS developers, we call these three categories permissive, weak copyleft, and strong copyleft (Almeida et al. 2019; Goldstein 2019; Sen et al. 2008).

Permissive licenses are often textually simple and short (e.g., MIT and BSD licenses) and the most basic type of open source licenses: they allow reusers to do whatever they want with the software as long as they abide by the notice requirements (Meeker 2017). Strong and weak copyleft licenses add restrictions to the permissive licenses. For instance, if you distribute binaries based on copyleft licensed software, you must make the source code for those binaries available. Also, the source code must be available under the same (or in some cases similar) copyleft terms under which you got the code, and you cannot place additional restrictions on the licensee's exercise of the license (Meeker 2017).

At the time of writing this dissertation, there are 104 Open Source Initiative (2020a) (OSI) approved licenses. However, 17 of these licenses have been super seeded or retired by the creator of the license so they should not be used to license any new code (Open Source Initiative 2020a). A complete list of categorized (Blue Oak Council 2020a,b; FOSSA 2020; nexB inc. 2020) OSI -approved licenses is found in Appendix 1. Even though the number of licenses is substantial, there are only 12 of them (listed in Table 1) that are considered by OSI as "popular and widely-used or with strong communities."

2.2.2 Permissive licenses

A permissive open source license permits derivative works without publishing the source code (i.e., proprietary software), along with the freedom to use, modify, and redistribute the software (Goldstein 2019). However, there are some typical restrictions or obligations in permissive licenses. First, the work is provided "as is." You may not hold the author liable of any damages caused by the use of software. This liability clause is found in almost all OSS licenses. Second, licenses often include a notice requirement. The recipient of OSS must be informed that certain OSS, which is available under the noticed license, is included in the software being delivered (Meeker 2017). Each open source license has its own notice requirements. Typically, these include providing entire copies of applicable licenses and acknowledgement of authors.

One representative example of a permissive license is the popular MIT license.

MIT License
Copyright (c) [year] [fullname]
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

As can be seen, the text of the license is fairly short and understandable, even for a non-lawyer. It explicitly states numerous rights and only includes a simple obligation of acknowledgement and a liability clause. However, all permissive licenses are not as simple and short as MIT or other academic licenses, such as different versions of the BSD license (Open Source Initiative 2020b,c). For example, the popular Apache 2.0 license (Apache Software Foundation 2020) is much more verbose than many other permissive licenses. As such, it is more difficult for an average software developer to understand. It also has two patent clauses that are not found in more simple permissive licenses. The first states that the software can be used without any obligations, regardless of the software patents that are in effect. The second states that if software users initiate litigation over patent infringement, they lose their license.

Permissive licenses offer reusers maximum freedom to use the software as they wish and even distribute the software under different license terms. However, this was not the intention of the founders of free software trying to preserve free access to the software and its code (Von Krogh and Von Hippel 2003). For this purpose, copyleft licenses were invented.

2.2.3 Copyleft licenses

The GNU General Public License (GPL), the first copyleft license, allows anyone to copy, modify, and redistribute any GPL-licensed program as long as all distributions include source code. It also sets another condition on redistributions: any program derived from a GPL-covered program must itself be distributed under the GPL. This condition, known as "copyleft," is a key legal innovation of the GPL (Stoltz 2005). According to the Free Software Foundation (2020c), the authors of the GPL license, "Copyleft is a method for making a software program free, while requiring that all modified and extended versions of the program also be free, and released under the same terms and conditions."

Even though GPL in its various versions and offshoots, including the Lesser General Public License (LGPL) and Affero General Public License (AGPL), are the most famous copyleft licenses, they are by no means the only ones. The Mozilla Public License (MPL), Common Development and Distribution License (CDDL), and the Eclipse Public License (EPL) are also well known. What is common to all copyleft licenses is that other developers have the right to use, modify, and share the work as long as the reciprocity obligation described in the license is maintained (Goldstein 2019). They also have notice requirements and liability clauses, which are also found in permissive licenses. What is varied between different copyleft licenses are the different levels of reciprocity characteristics.

There are two types of copyleft licenses: strong and weak. In the case of strong copyleft (e.g., GPL), when using a piece of software covered by a copyleft license and another piece of software covered by some other license, combining the two pieces of code in a single work must, by law, result in a work available under the terms of the copyleft license. Strong copyleft licenses are often used in applications or other individual pieces of software such as GNU/Linux operating system (which uses GNU packages on top of a Linux kernel). Whereas strong copyleft licenses have strong reciprocity, weak copyleft licenses allow other software to use the copyleft-licensed software and be redistributed without the requirement for the software to also be copyleft licensed. Weak copyleft licenses are often used to create software libraries or to allow proprietary plug-ins to extend the weak copyleft-licensed software. Only changes to the weak-copyleft-licensed software (e.g., the library or individual source file) itself can become subject to the copyleft provisions of such a license. The most popular weak copyleft licenses are LGPL, MPL, CDDL, and EPL.

The phrases in which the copyleft is stated in the actual license differ. GPLv2 uses the phrase "derivative work," which was identified to be problematic as the legal question of when two interacting programs form a derivative work is by

no means simple (Hazen 1986) and will determine how broadly the GPL applies. For this reason, version 3 uses "work based on the program" making the license more clear under international copyright laws (Free Software Foundation et al. 2013). In comparison, MPLv2 uses phrases"covered software" and "larger work," making it clear that reciprocity characters apply to specific files instead of the work as a whole.

Also, the concept of distribution matters because the requirements of many open source licenses are triggered only when the software is distributed, which is defined as transferring a copy of a copyrighted work (such as software) from one legal person to another (Meeker 2017). As the Free Software Foundation (2020c) states, "To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code, or any program derived from it, but only if the distribution terms are unchanged." Because license terms are mostly triggered only when software is distributed, a person who does not distribute software cannot violate an open source license's terms. And because "legal person" includes a corporation, there is no distribution, and therefore no risk of violating a license's terms, if software is merely transferred between employees of the same company (Meeker 2017).

If a software application is used over the Internet, it is usually not qualified as distribution of that software (Meeker 2017). This creates a distinguished loophole in the ordinary GPL, where the copyleft provisions are not triggered if the software is simply used but not distributed. This loophole can be used by application service providers to create derivative works of GPL-licensed code and offer application services over the network without making the changes in the code available to the public. AGPL (Free Software Foundation 2007) was designed to close this loophole, as stated in the version 3 of the license, "The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public. The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version."

A common concern among software companies is that by reusing code licensed under GPL (or similar copyleft license) as part of their proprietary code, the company's code will be "infected" or "contaminated" and must be licensed under GPL or forced into the public domain. This concern discourages some from using GPL code, and for this reason, copyleft licenses are sometimes called viral licenses, even though usage of this term should be avoided (Rosen 2001). According to Meeker (2017), concerns related to the consequences of illegal reuse of GPL are largely unfounded. It is clear that if a developer combines GPL code with incompatible (e.g., commercial) code and redistributes that combination, it violates the GPL. Even though the author of the violated GPL code can exercise

their right to bring a claim to copyright infringement, the copyright law does not enforce the license offending code under GPL. The remedy for copyright infringement is then either money (damages) or stop using the code (injunction) (Meeker 2017). Therefore, combining GPL code with other incompatible code does not "infect" the incompatible code or convert it into GPL code.

Out of all the common weak copyleft licenses, LGPL has the strongest reciprocity characteristics. According to the Free Software Foundation (2020a) on the GNU Licensing FAQ, "(1) If you statically link against an LGPL licensed library, you must also provide your application in an object (not necessarily source) format, so that a user has the opportunity to modify the library and relink the application. (2) If you dynamically link against an LGPLed library already present on the user's computer, you need not convey the library's source. On the other hand, if you yourself convey the executable LGPLed library along with your application, whether linked with statically or dynamically, you must also convey the library's sources, in one of the ways for which the LGPL provides." However, this strong interpretation has been challenged by Rosen (2005) as he does not consider linking as a means to create derivative work.

The Mozilla Foundation and their Firefox browser uses mostly MPLv2.0, whereas Eclipse foundation's software (such as Eclipse IDE or Eclipse Glassfish) uses mostly EPLv2.0. According to the Mozilla (2020) FAQ, the "MPL fills a useful space in the spectrum of free and open source software licenses, sitting between the Apache license, which does not require modifications to be shared, and the GNU family of licenses, which requires modifications to be shared under a much broader set of circumstances than the MPL. The MPL's 'file-level' copyleft is designed to encourage contributors to share modifications they make to your code, while still allowing them to combine your code with code under other licenses (open or proprietary) with minimal restrictions." EPL has similar characteristics and was developed for the needs of the Eclipse Foundation. EPL was originally derived from Common Public License version 1.0.

CDDL is based on MPLv1.1, but it has made few changes to make it more accessible to developers, and since the MPLv2.0 had significant modifications from version 1.1., further separation between these licenses is apparent. The two main differences between the two licenses are GPL compatibility and simplicity: the MPLv2.0 is compatible with the GPL, while the CDDL is not. Some argue that CDDL uses a simpler, more consistent language and that it is better structured to make the license more understandable and increase developer's adoption rate (Sass 2015). For example, Netbeans uses CDDL.

2.2.4 License compatibility

Licenses that can be used together because their limitations, conditions, and permissions do not contradict with each other are considered compatible. However, some OSS licenses are not compatible with each other, and this can make it legally impossible to mix (or link) open source code if the components have different licenses (German and Hassan 2009). This incompatibility occurs when following

one license's terms violates another license's terms.

Permissive licenses are typically compatible with other licenses in the sense that software with a permissive license can be combined as part of other permissive-, copyleft-, or commercial-licensed software. However, as described by the Free Software Foundation (2020b), one notable exception is Apache 2.0, which is not compatible with GPLv2 because it has patent-related restrictions that are not in that GPL version. As it is stated in GPLv2 license, "You may not impose any further restrictions on the recipients' exercise of the rights granted herein," making these two licenses incompatible. However, from the Free Software Foundation's point of view, the patent termination provision is a good thing, which is why they recommend the Apache 2.0 license for substantial programs over other simple permissive licenses (Free Software Foundation 2020b). This incompatibility is fixed in GPL version 3.

Permissive licenses are usually also compatible the other way around: adding copyleft-licensed software as part of permissive-licensed software is permitted. However, in this case, permissive-licensed software must often be relicensed under the copyleft license.

Incompatibilities between copyleft licenses are more common. The Free Software Foundation (2020b) lists more than 40 free software licenses that are incompatible with GPL. However, many of these are not approved open source licenses as listed by the Open Source Initiative (2020a). The incompatibility list includes, for example, Eclipse Public License Version 2.0 and Common Development and Distribution License 1.0.

Licensor and licensees have created ingenious solutions to address the incompatibility problem, as described by German and Hassan (2009). For example, Mozilla code, originally licensed under MPLv1.1, could not be combined with code under the incompatible GPLv2. To solve this problem, the Mozilla Foundation chose to relicense Mozilla under a "Disjunctive Licensing," that is, under three licenses (the GPLv2, LGPLv2.1, and MPLv1.1) and let the licensee choose one of them. However, later versions of Mozilla have been further relicensed under MPLv2.0 as it is compatible with (L)GPL licenses. Another example to solve compatibility problem is the "exception," where the licensor adds an addendum to a license (such as the GPL) that permits certain uses that would otherwise be forbidden. One example is the GPLv3 license of the GNU Compiler Collection (GCC) runtime library, which has a following exception (Free Software Foundation 2009) : "You have permission to propagate a work of Target Code formed by combining the Runtime Library with Independent Modules, even if such propagation would otherwise violate the terms of GPLv3, provided that all Target Code was generated by Eligible Compilation Processes. You may then convey such a combination under terms of your choice, consistent with the licensing of the Independent Modules." Because of this exception, a program compiled using GCC can be distributed under any license (open source or not).

Compatibility between licenses is not only limited to if certain pieces of software are used as part of another program. Various forms of program linking are used to create a combined program (Rosen 2001). For example, according to the

Free Software Foundation (2020b), linking LGPLed code statically to incompatibly licensed software is not allowed but dynamic linking is. This creates another dimension to license compatibility, where the build instructions must be taken into account when determining license compatibility.

Adoption of software licenses are also dictated by the restrictions and guidelines of open source communities (Vendome et al. 2018). For example, the Apache Software Foundation (ASF) requires contributions to be licensed under the Apache 2.0 License. This makes otherwise compatible GPL 3 license incompatible in ASF software, because it would require Apache-licensed software to be relicensed under GPL (Apache Software Foundation 2019). Debian uses the "Debian Social Contract" or "Debian Free Software Guidelines" (DFSG) (Debian 2004) to evaluate whether a license is free or non-free and whether it can be bundled into Debian or must be distributed separately.

2.3 Summary

In this chapter, we have described intellectual property rights and open source licenses. IPRs protect the author(s) of software from the unauthorized exploitation of their work. OSS licenses give rights to, for example, use, copy and create derivative works and redistribute the copyright protected software. Next, we will present and open source license compliance, and related process that ensures that reusers of OSS follow the terms of open source licenses.

3 OPEN SOURCE LICENSE COMPLIANCE

The reuse of open source introduces specific IPR-related risks that are an extension of traditional software project risks such as budget overruns, unsatisfactory functionality, quality, and maintainability (Boehm 1989). To mitigate the risks related to OSS reuse, the reuse must be governed (Harutyunyan et al. 2019). OSS governance refers to the set of processes, best practices, and tools to reuse OSS components while minimizing their risks and maximizing their benefit (Harutyunyan et al. 2019). Open source license compliance is a part of the OSS governance. In simple terms, open source license compliance means that the users of OSS must satisfy the license obligations for the OSS they use (Haddad 2019).

3.1 Overview

Clearly, before the reuse can be considered, the candidate software must first be found. As the amount of OSS today is substantial, it may be difficult to find appropriate reuse code candidates using traditional search engines such as Google[1]. For instance, engines such as Sourcerer by Bajracharya et al. (2006) and Maracatu by Garcia et al. (2006) can be used to find source code. Sourcerer is capable of searching both OSS implementations and their usage, as well as program structures from the Internet, whereas Maracatu is used for retrieving source code components from development repositories by combining text mining and facet-based searches. At the time of writing this dissertation, however, there is very limited availability of online code search tools that are specifically used for finding OSS code. Only notable search engine we were able to find was searchcode[2]. Also, the component repositories such as GitHub offer their own search tools.

The efficient reuse of OSS in companies and other organizations relies on governance processes and best practices. These include: establishing, communicating, adjusting, and improving reuse policy and process; creating, updating,

[1] https://google.com accessed Jan 18, 2021

[2] https://searchcode.com/ accessed Jan 18, 2021

maintaining, using, and auditing a component repository; using tools to create, update, maintain, and search a component repository; providing all relevant metadata for components; tracking the OSS reuse and providing its prior approval data; and adding security check information to components in the repository (Harutyunyan and Riehle 2019b). In addition, Haddad (2019) also lists recommended practices such as setting up a review board, setting up automated systems to detect OSS, improving sourcing practices to make suppliers comply with OSS licenses, scaling up legal support, and creating checkpoints and checklists for compliance activities.

Harutyunyan et al. (2019) describe the industry requirements for OSS governance tools to facilitate the use of OSS in commercial products. They identified five key categories of OSS governance tool requirements: 1) the tracking and reuse of OSS components, 2) license compliance of OSS components, 3) search and selection of OSS components, 4) architecture model for software products, and 5) other requirements (security, export restrictions, etc.).

The processes, tools, and best practices described above form the large-scale picture of OSS reuse. Open source license compliance serves a key role in this picture as it ensures that IPR-related risks are mitigated. Open source license compliance can be addressed through a three-step process of *identify* the potentially reused software and its licenses, *approve* the usage, and *satisfy* the license obligations (Haddad 2019). The goal of the identify step is to identify all OSS (packages and snippets), their origin, licenses, and any licensing inconsistencies. The second step uses this collected information to make a decision whether the OSS or parts of it can be reused. The final step makes sure that the obligations listed in the licenses of reused software are fulfilled: source code, copyright notices, and license texts are made available if necessary.

The complexity of open source license compliance comes from the different restrictions and obligations of the OSS licenses compared with commercial licenses, where the obligations typically consist only of the payment for the right to use or distribute the software (Schoettle 2019). Also, the incompatibility of some OSS licenses and the fact that the overall license of a product might be different than the license of each of its files can further complicate compliance (German et al. 2010b). Nevertheless, ensuring license compliance before the software is distributed is often neglected (Schoettle 2019). For example, Cisco and VMWare have faced legal action for violating the licensing terms of the Linux kernel (Ryan 2009; Vaughan-Nichols 2015).

Another aspect of open source license compliance is that companies integrating OSS into their commercial products, software solutions or services, also want to protect their own intellectual property from unintended disclosure. Under some licenses (e.g., GPLv3), the contributor of the software must also grant a patent license to the contents of the contributor version (Lau and Ker 2020). In this way, patent holders may accidentally grant rights to, for example, use, sell, and import their patented technology.

The risk of unnoticed noncompliance introduced to a software is high since software components and code snippets are commonly reused (Mikkonen and

Taivalsaari 2019). Component reuse often occurs when using package managers, such as npm[3] or Maven[4], or by using source code repositories such as GitHub. The npm and Maven packages include the license information as part of their metadata, whereas GitHub projects may or may not contain licensing information. Through an examination of 1,692,135 code repositories in GitHub, only 14.9% of them specify a license, making their reuse from a licensing perspective a challenge (Kapitsaki et al. 2015). In addition to reusing software components as a whole, snippets of code are often reused. Developers routinely search for solutions for all kinds of programming challenges from forums such as Stack Overflow[5] and use these as a part of their software (Mikkonen and Taivalsaari 2019).

Vendome et al. (2018) describe the licensing issues that developers generally face when reusing OSS and refer to these as licensing bugs. These bugs are related to issues such as potential licensing violations, licensing content-related issues, and the breach of guidelines that can prevent software from being distributed or modified, for example, by preventing a patch from being accepted (Mathur et al. 2012; Vendome et al. 2018). Fixing these bugs may require substantial effort. The violating code needs to be replaced or the software's license needs to be migrated to be compatible with the reused code (Vendome et al. 2018).

Distributing software with licensing violations leads to illegally distributing copyrighted material. When a licensing (or other IPR) violation is created in a software package, it poses a threat individuals besides the creator of such violation. When a developer (even acting in good faith) adds a contribution, that infringe on the IPR holder's copyright or patent, every subsequent developer can be liable according to copyright and patent laws, even if they did not know that the software infringes a third-party right (Välimäki and Oksanen 2005).

3.2 Identifying used open source software and its licenses

The core of the open source license compliance effort is to identify open source code and their respective licenses, which form the first step of the compliance process (Haddad 2019). To store the results of a license analysis and to fulfill the foregoing information and documentation requirements, one central document or repository of the bill of materials should be maintained (Schoettle 2019). This bill of material includes all necessary texts, notes, and information and are contained in a structured manner. The information lists the OSS used, its origin, license of each source file, and the overall license(s) of the reused code. Rigorous maintenance of a bill of materials is especially important in software projects that reuse substantial amount of OSS, since tracking the reuse afterwards is difficult, especially when reusing snippets of code.

[3] https://www.npmjs.com/ accessed Jan 18, 2021
[4] http://maven.apache.org/ accessed Jan 18, 2021
[5] https://stackoverflow.com accessed Jan 18, 2021

First, the true origin of the potentially reused software should be ensured. A reused package or snippet may not be fully written by the authors since it may contain copied code, as code cloning is very common way of reusing source code in software development (Ain et al. 2019). Cloning becomes especially problematic when it violates the license terms or IPRs of the original author. There are several examples where GPL-licensed code has been copied in the ways that violates the terms of the GPL license (Duan et al. 2017; Feng et al. 2019). Also, it has been shown that copying between OSS projects and developer forums is common (An et al. 2017; Ragkhitwetsagul et al. 2018). Numerous tools and methods have been developed to detect code clones that assist in identifying copied code (Ain et al. 2019). However, identifying the true origin of the source code is very challenging, because it is hard to build a universal repository that contains every version of every open source repository artifact ever released (German and Di Penta 2012).

The second part of the identification step, and probably the most determinant part for dealing with license compliance (Kapitsaki et al. 2015), is the reliable identification of the software's license(s). The license of an open source program can be indicated in different ways within the actual software package (Tuunanen et al. 2009). The license of a source code file is typically specified in a comment at the very beginning of the file. German et al. (2010b) refer to this region of the file as its license statement. A license statement typically contains four sections: 1) a list of copyright owners, 2) a list of authors (if different from the copyright owners), 3) the license or licenses that cover the file, and 4) warranty and liability statements. The licenses in the licensing statement can be of two types (German et al. 2010b):

by-inclusion: the text of the license is embedded in the file, and here, examples are the BSD and the MIT families of licenses;

by-reference: the license statement indicates where the text of the license can be found (file or url). This technique is usually used for open source licenses such as GPL, Apache, and MPL which have a longer license text.

There are several factors that make identifying the origin and license of OSS a challenge. For instance, ensuring OSS license compliance is not only relevant regarding the license conditions of directly used OSS, but it also covers those OSS, that directly used OSS depends on (Dyck et al. 2018). When a reused package depends on other packages, the total amount of reused code can be substantial (Mikkonen and Taivalsaari 2019) and, thus, impractical, or even impossible, to inspect manually.

While all source files of OSS can have the same license, it is common that there are several (possible incompatible) licenses used, causing a license-mismatch problem (German and Hassan 2009). For these packages, license identification of the whole component is not straightforward and may need further analysis in the approval step.

Vendome et al. (2018) list several licensing content-related bugs that make the identification a challenge, such as incorrect licensing, license inconsistencies,

missing licensing, license textual issues, and outdated licensing. Due to these issues, packages license information could be misrepresented or incomplete. Fortunately, fixing these licensing bugs is relatively simple, since the actual system does not need to be modified (Vendome et al. 2018). It is adequate to add the proper annotation, move the license file, or add license headers. German et al. (2010b) present some examples of OSS packages with these kind of licensing issues. One is an application, where eight files were identified under the LGPLv2+, even though this license does not exist. Other MIT-style licensed application had three files out of 37 that contained different liability and warranty clauses than the rest.

Reused snippets are often copied without proper identification of their license or copyright holder, making the identification of their license challenging and also resulting potential violations of their license terms (Romansky et al. 2018). Also, if these snippets do not include the copyright notice, then they violate the author's right to claim authorship.

When OSS is reused as a package, the package usually contains metadata describing the license of the package. However, this declared license of the package and license of the actual source files may differ (German et al. 2010a; Kapitsaki et al. 2017; Manabe et al. 2014).

To get a reliable list of the used licenses, the source code of the reused package must be scanned, identifying the licenses of each source file. Also, other sources for the license information, such as the binary of the OSS and the project website might be used (Dyck et al. 2016), when the license is not explicitly stated.

3.3 Approving the reuse of OSS

After the reused software and its licenses have been identified, the reuse of the package must be approved. The approval step ensures that no OSS is reused when licensing terms are violated or non-approved licenses are used. However, to approve the reuse of a specific OSS, the results of the previous step need to be presented in a meaningful way, and the terms of the used licenses must be understood.

If the potentially reused package contains licensing violations, it can not be approved. Licensing violations consist of license incompatibility or other types of violations (Vendome et al. 2018). The occurrence of a license-mismatch problem in OSS code may be an indication of license incompatibility issues or other serious violations or can also be an indication of less severe issues or no issues at all. A typical example of software having multiple licenses is a case where some files are licensed under copyleft license and some other under permissive license(s). This combination is typically allowed, and reuse can be approved as long as the licensing terms of all licenses are followed. Combining the software of conflicting licenses is typically not allowed, and these licensing incompatibilities must be identified and removed before the reuse of OSS can be approved. An

example of such incompatibility was described by German et al. (2010b), where an application according to its web site, was licensed under the GPLv2. The majority of its files were licensed under the GPLv2+, but three of them were under the GPLv3+. Licensing conflict was caused by the GPLv3+ files: a file licensed under the GPLv3+ cannot be combined into a system under the GPLv2.

However, license compatibility checking must also consider the way in which the code is used, as the technical means of combining the code may affect the compatibility. Hammouda et al. (2010) present some architectural design decisions motivated by the legal concerns associated with open source licensing issues. These open source legality patterns, that is, recurring design decisions, are created to simplify and mitigate the risks of combining differently licensed OSS software. When these patterns are used, even incompatible licensed OSS can be combined and reused. Patterns include, for example, standardized interfaces, dynamic linking, and data-driven communication (Hammouda et al. 2010).

Other types of licensing violations are serious violations that do not fall under license incompatibility. For example, these include illegal copying of source code. Vendome et al. (2018) introduce an example of illegitimate copying, where a developer copied MIT-licensed code and removed the original author from the copyright statement of the file(s). Also, using snippets from developer forums may result in licensing violations in case they are considered "original works", that is, they are original enough to be copyrightable. For instance, code examples on Stack Overflow are governed by the Creative Commons Attribute-ShareAlike 3.0 Unported license (Stack Exchange Inc. 2020), which means they require giving appropriate credit, providing a link to the license, and indicating the changes that were made (Creative Commons 2020).

Even though the licensing content-related issues identified in the previous step (e.g., license inconsistencies or missing licensing) may not violate intellectual properties, they may prevent the approval of software distribution. This is due to the fact that some communities (e.g., Fedora (2013a)) require that package metadata must include correct license information before they can list and distribute the software.

The litigious nature of the language used in OSS licenses can make understanding the ramifications of these licenses difficult. Almeida et al. (2019) conduct a survey that posed development scenarios involving three popular open source licenses (GPLv3, LGPLv3, and MPLv2) both alone and in combination. The survey included 375 respondents, who were largely developers. Their answers were consistent with those of a legal expert's opinion in 62% of 42 cases. Even though developers understood cases that involved one license, they had difficulties when multiple licenses were involved. Also, 36% of the participants reported using resources such as Wikipedia, https://tldrlegal.com and https://choosealicense.com. Even though developers had some understanding of the licenses, only three well-known licenses were used in the survey. The difficulty in understanding licenses can prevent reusers from using certain licenses. This is due to the uncertainty of the implications of such a decision. The adoption of a license can be prevented, for example, due to the following reasons: the problem of understanding whether

the new license is compatible with the licenses of the software's dependencies, the misunderstanding of the implications of a clause, or the uncertainty whether a license satisfies the business model of the developers (Vendome et al. 2018).

The communities maintained by foundations and commercial organisations (e.g., the Apache Software Foundation, Debian, the Eclipse Software Foundation or Fedora), may especially perceive the use of different kinds of conditions as a necessary means for avoiding legal disputes and for their own legal protection (Gamalielsson and Lundell 2017). The rules and restrictions for the approval of software in such communities originate from how a particular community defines the acceptance of software, which can be stricter than utilizing an open source license, and how the community interprets licenses based upon their policies (Vendome et al. 2018). Examples of such acceptance criteria include Debian free software guidelines (DFSG) (Debian 2004) and Fedora licensing guide (Fedora 2013b). In these cases, software with non-approve licenses, including even well known OSS licenses, cannot be included in the main distribution and must be distributed separately (Vendome et al. 2018). Unlike licensing bugs or the incompatibility of licenses as such, these reflect how a community enforces a particular licensing policy.

The runtime environment also affects whether the OSS can be approved. For example, GPLv3.0 requires that it must be possible to install modified versions of the OSS on devices where the original version is running. This practically prevents GPLv3.0 licensed code on digital-rights-management-protected environments such as iPhones (Schoettle 2019).

Some licenses (e.g., GPLv3.0 and Apache 2.0) address issues related to patents, including their litigation. For example, engaging in a patent lawsuit with work that is derivative or reuses source/binaries under these licenses will invalidate the license. For these reasons, the reuse of OSS licensed under these licenses may be rejected.

Companies use trademarks to protect their name, logo, products and promotions (Fosfuri et al. 2008). Trademark-related restrictions (i.e., preventing someone from using the same product name or logo) can prove problematic within an open source community and may prevent the approval of OSS. Organizations want to provide a quality guarantee, but communities like Debian debate whether these restrictions are in-line with their guidelines that promote the freeness of OSS (Vendome et al. 2018).

The approval step also includes questions related to laws and their interpretations, such as the following (Vendome et al. 2018): What is copyrightable? What is a derivative work? How licenses are interpreted under different jurisdictions? It has been demonstrated that copying as few as 27 lines of code from 525,000 can constitute copyright infringement when the code is crucial and the incorporating software would not function without it (Mertzel 2008). As it is very common to copy code snippets from and to developer forums such as Stack Overflow, the question whether these snippets are copyrightable becomes relevant (Romansky et al. 2018). If such snippet is copyrightable, the violation of its license terms are very likely. Also, to be able to distribute a derivative work, its creator needs a

license from the work it is based on. Therefore, the question of what constitutes a derivative work is critical (Vendome et al. 2018). Some systems, for example, GNU (Free Software Foundation 2020a), provide clarifications on whether something is considered a derivative work of another. Also, this issue has been debated among lawyers extensively over the years (Determann 2006; Evans and Layne-Farrar 2004; Hazen 1986; Stoltz 2005). However, since the scope and interpretations of copyright law (similar to patent and trademarks) can only be truly addressed by lawyers, which we are not, these questions fall outside the scope of the current study.

3.4 Satisfying OSS license obligations

As reuse of OSS is approved and is ready to be reused, the license obligations must be fulfilled. These obligations come into effect when software is distributed or made available over the network (in case of network reciprocal licenses such as AGPL). To fulfill these obligations, the bill of materials that is created in the first step of license compliance process is vital as it lists all used OSS and its licenses. Open source licenses require that the distributor of the OSS must do the following (Haddad 2018):

- In case of distributed software is licensed under a copyleft-license, inform the end user how to obtain a copy of the source code.
- Acknowledge the use of open source by providing the required copyright, attribution, and license notices for all applicable OSS (components and snippets).
- Reproduce the entire license text for the open source code included in the product.

This information is collected in a report that is usually called *open source notices* or *attribution statements* (Goldstein 2020a). It includes all necessary information to fulfill the license obligations.

OSS may be used "as is" or in a modified form. In either case, the source code must be available if the software is licensed under copyleft license. For this reason, copyleft licensed source code must be prepared for redistribution. This may include the source code of the entire software (in the case of strong copyleft), or, for example, individual files (in case of weak, file level, copyleft). Typically, the most simple solution is to provide a link in the *open source notices* to a public web page that hosts the source code, regardless of if the source has been modified or not.

Acknowledgement requirements consist of three parts: reproduction of copyright texts, attribution notices, and license notices. Open source licenses almost always require reproduction of copyright statements, which are included in the source files. The license notice informs the user that the distributed software includes software under the given license. Attribution statements, which give

credit to, or attributes, the creator of the work whom the code is borrowed from, are explicitly required by some OSS licenses such as Apache 2.0.

Gathering the copyright information manually is very time-consuming and error-prone (Dyck et al. 2018). For this reason, any effort that makes this process easier is preferable. For example, the Linux Foundation (Winslow 2020) recommend using a more general statement in a form similar to the following (where XYZ is the project's name):

- Copyright The XYZ Authors.
- Copyright The XYZ Contributors.
- Copyright Contributors to the XYZ project.

As Winslow (2020) states, "These statements are intended to communicate the following: (i) the work is copyrighted, (ii) the contributors of the code licensed it, but retain ownership of their copyrights; and (iii) it was licensed for distribution as part of the named project. By using a common format, the projects avoid having to maintain lists of names of the authors or copyright holders, years or ranges of years, and variations on the (c) symbol. This aims to minimize the burden on developers and maintainers as well as redistributors of the code, particularly where compliance with the license requires that further distributions retain or reproduce copyright notices."

The license notice is typically very simple, as it only requires that it is made apparent the work contains software under a given license. However, the reproduction of the full license text is usually required. Some licenses are explicit in their attribution requirements. Apache 2.0 requires attribution statements when a file called NOTICE exists. However, open source attribution obligations, as specified in the many other common licenses, are usually very simply stated and are subject to a great deal of interpretation regarding what is legally required and what is the best way to meet these obligations (Clark 2015).

Clark (2015) describes *open source notices* examples of two major software providers, Mozilla and Google, which provide examples of very different approaches. Both these examples do not only fulfill the minimum legal requirements, but also the spirit of the attribution where the developers deserve credit for their contribution. Whenever OSS is reused, Firefox and Chrome can be used as examples of fair and obligation-fulfilling *open source notices*. In Mozilla Firefox, by clicking on "About Firefox" in the Help menu and then by clicking the "Licensing Information" link, this takes you to an "about:license" page. The information on the page lists the names of the licenses used in Firefox, and each license name is a link that takes you to the complementary details that include the path name of the component subject to that license, the copyright statement, and the texts of the licenses and notices (see Figure 2).

In Google Chrome, by clicking on "About Google Chrome" and then by clicking the "open source software" link, this takes you to a "chrome://credits/" page. The page lists the open source projects used in Chrome. Each project item provides links to the project home page and the project license (see Figure 3).

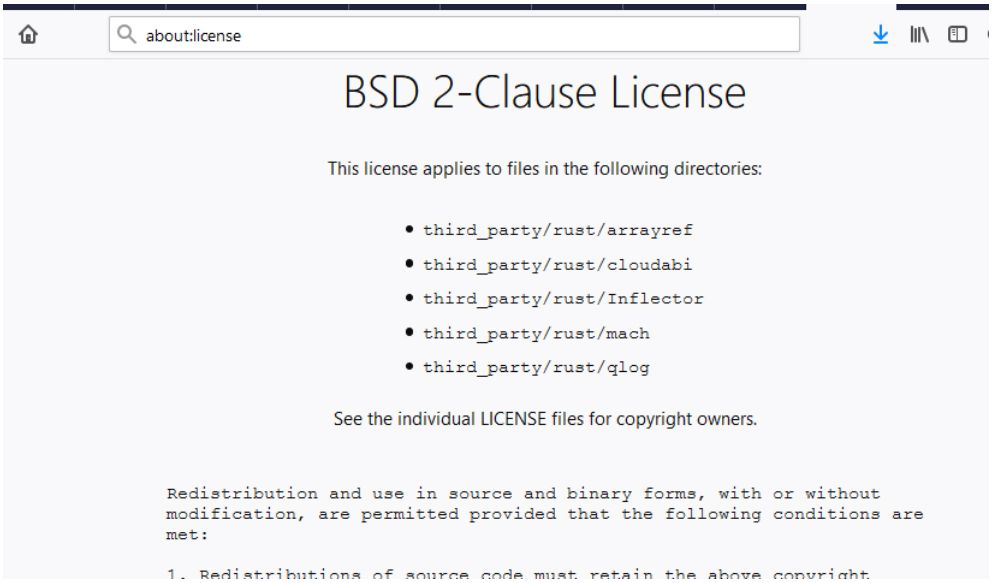

FIGURE 2 Mozilla BSD 2-Clause attribution.

FIGURE 3 Chrome attribution page.

3.5 Summary

In this chapter we have described the open source license compliance that can be addressed through a three-step process where the user must identify, approve, and satisfy the reuse of OSS. This was preceded by the introduction of IPRs and open source licenses. These form the theoretical background of the current study. Next, we will present the results of our study and answer our research questions.

4 RESULTS FROM THE DESIGN CYCLE

This chapter presents our reverse engineering approach for the automated support of OSS license compliance, its implementation via a tool called ASLA (Automated Software License Analyzer), and tool evaluation. This chapter will address the research questions for the design cycles part:

RQ1 What are the user needs to fulfill automated open source license compliance?
RQ2 What software features are needed to fulfill the user needs?

The need for automated OSS license compliance was discovered in the industry that reuse OSS as part of their commercial products. Based on the initial industry needs automated license compliance was also identified to be a relevant academic problem that extended to the development of ASLA. User needs of ASLA are based on real-life needs present in the industry at the time. The compatibility rules between the licenses (see Figure 10), that are used as an example in this chapter, are based on the views of industry lawyers at the time as well as on the views of well known OSS license expert, lawyer Lawrence Rosen (2005).

Our goal was to support IPR-aware reuse of OSS packages through an automated license analysis by retrieving software license information from source code modules. It was motivated by a typical problem in OSS development: license compliance should be followed, but reliable and detailed manual license information retrieval and analysis is time-consuming and error-prone and requires a profound knowledge of OSS licenses.

The research of the first cycle followed the Design Science Research (DSR) approach, by specifying, implementing, and testing an artifact of OSS reuse utilization based on a license analysis. DSR is composed of the three related cycles that should not be confused with the design and review cycles of this study. DSR includes (i) the relevance cycle, (ii) the rigor cycle, and (iii) the design cycle. The DSR relevance cycle ensures that technical solutions solve practical business problems and address the corresponding opportunities. The DSR rigor cycle connects the prior scientific knowledge and theories with the research (Hevner 2007;

Iivari 2007), also ensuring that appropriate methods are applied in the construction and evaluation of the design artifact (Venable 2010). The DSR design cycle includes the construction and evaluation phase of the artifact. Peffers et al. (2007) present a more structured composition of DSR, as follows: (i) identify problem, (ii) define solution objectives, (iii) design and development, (iv) demonstration, (v) evaluation, and (vi) communication.

ASLA was initially introduced in Tuunanen et al. (2006a), which describes the design and realization of license identification and dependency analysis. Tuunanen et al. (2006b) extend the previous paper, especially by addressing the issue of license retrieval from an OSS perspective and by providing a more detailed description of ASLA. After further development, the architecture and functionality of ASLA was described in a more detailed manner, and an extended evaluation of the tool has been published in Tuunanen et al. (2009).

At the time when our articles were written, there were no license analysis tools available in an academic sense. However, some open source solutions existed, such as OSLC version 2.0[1] and FOSSology version 1.0[2]. The functionality and performance of these tools are evaluated in Tuunanen et al. (2009) underlining the research gap (DSR step i) that ASLA fulfilled. There was especially a clear need for more efficient license identification from source files and an automated compatibility analysis. As we applied the DSR approach in the development of ASLA, we identified the typical problems in the OSS license compliance process, resulting in the user needs for the automated license analysis software (DSR step ii).

System architecture and tool features were developed (DSR step iii) using an iterative process model as benefits of these models were well known at the time of the development (Larman and Basili 2003). The latest version of the ASLA system was implemented using Java programming language (version 1.6.0_03). The system architecture is presented in Figure 4 (Tuunanen et al. 2009). The demonstration and evaluation of the artifacts (DSR steps iv and v) resulted in the tool evaluation in terms of available features and in terms of identifying licenses compared with the other license analyzers mentioned above. The publications listed above fulfill the DSR step of communicating results in publications (DSR step vi).

In the following sections, the identified user needs that address research question RQ1.1 (see Table 2) are marked with Nx (N1 - N14). The features that address research question RQ2.1 (see Table 3) and fulfill user needs are marked with Fx (F1.1 - F9.1). The needs and features related to license identification and the dependency analysis are part of the identification step of the license compliance process, whereas the compatibility analysis and presentation of the identification results are part of the approval step. The needs and features related to the satisfy step of the compliance process include summary information and browsing of the results. A summary of user needs and features of ASLA and their relations are shown in Figure 5 (Tuunanen et al. 2009).

---

1 https://sourceforge.net/projects/oslc/files/oslc 202/oslc-2.0-stable/ accessed Jan 18, 2021

2 https://www.fossology.org/ accessed Jan 18, 2021

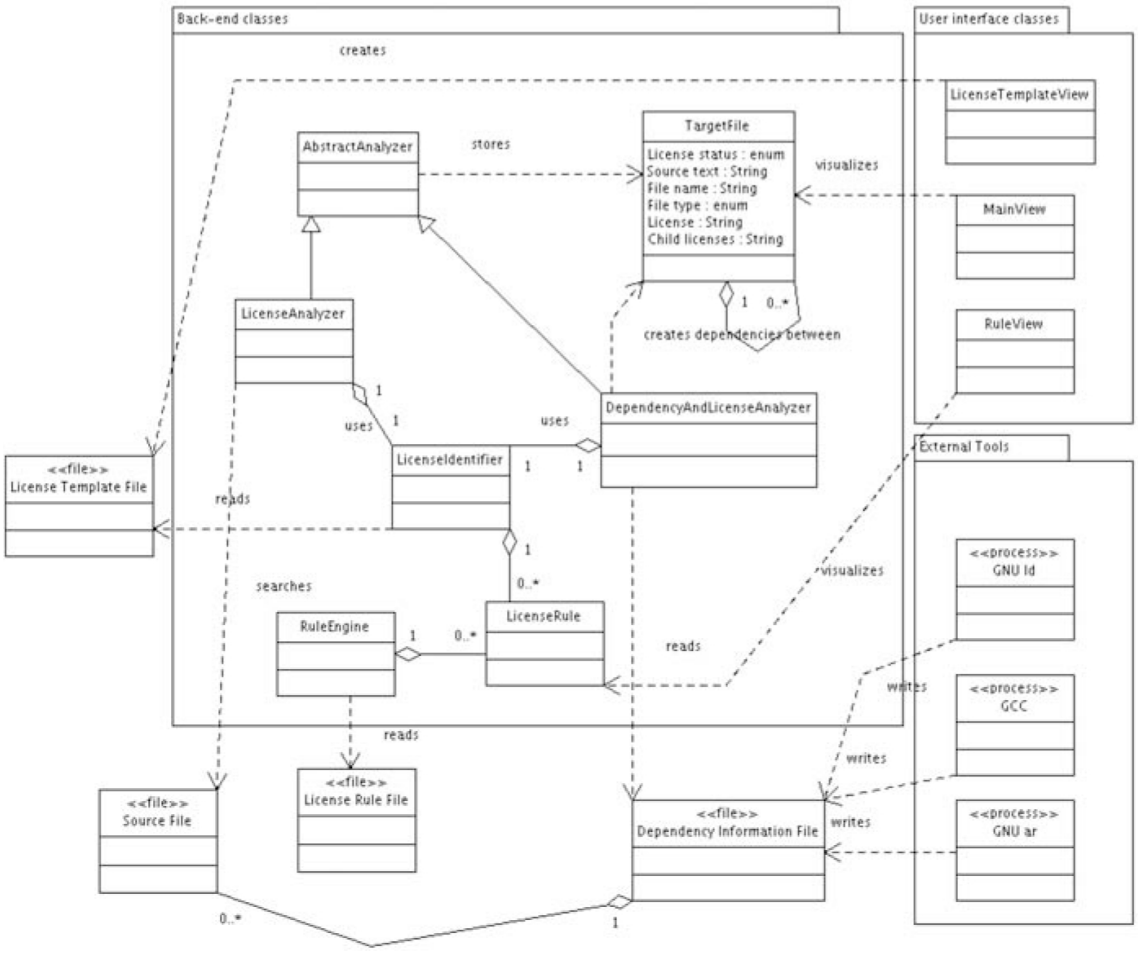

FIGURE 4 ASLA architecture (Tuunanen et al. 2009).

TABLE 2 Identified user needs for ASLA (Tuunanen et al. 2009).

User need		Rationale	Step*
N1	Identification of the program modules that are included in the program within a particular environment.	Get detailed list of used source files that will be analyzed and potentially reused.	I
N2	License identification of each source file.	Essential for comprehensive license analysis.	I
N3	Dependency identification of program modules.	Precondition for a detailed compatibility analysis.	I
N4	The formation of license compatibility rules.	Needed to detect license incompatibility issues.	A
N5	The identification of licensing problems.	Fundamental feature of the license analyzer as bugs or lack of them can dictate whether the component can be reused or not.	A
N6	The visualization of license analysis results.	To make an educated decision of the acceptance of the reused software and to display potential licensing bugs.	A
N7	Browsing of the results of the license analysis.	To see the list of used licenses and to identify potentially problematic components or source code files.	A,S
N8	Manual determination of the source code license.	Needed as automated license identification cannot be achieved for every file.	I
N9	The addition of license identification templates.	To handle previously unknown ways of indicating used license.	I
N10	The visualization of license compatibility rules.	As compatibility of the OSS licenses can be interpreted differently, the user should be aware of the used rules.	A
N11	The definition of license compatibility rules.	As compatibility of the OSS licenses can be interpreted differently, the user should be able to define them.	A
N12	Statistical information on the analyzed software package and the license analysis.	Assists in satisfying the license obligations.	S

* Refers to step of the license compliance process: identify (I), approve (A), and satisfy (S).

TABLE 3 Features of ASLA to satisfy the user's needs in Table 2 (Tuunanen et al. 2009).

Feature		Rationale	Step*
F1.1	Dependency and license analysis.	System-level feature providing dependency and license identification and compatibility analysis.	I
F1.2	License identification of source code files.	System-level feature providing license identification of each source file.	I
F2.1	Creation of dependency map.	To form a structure where dependencies of all program parts (source and binary files) are included to make a compatibility analysis.	I
F2.2	Separation of the files used in a specific environment.	Identify which files are used and which files are left out in selected environment.	I
F3.1	Automated identification of licenses in source code files.	Core feature, which is achieved by using license templates given as regular expressions.	I
F4.1	Automated license compatibility checking.	Checks for incompatibilities between licenses of the analyzed OSS component.	A
F4.2	Formation of license compatibility rules.	Rules define how two licenses cooperate with each other.	A
F5.1	Manual license determination of individual files.	Allows the tool user to set license of a file manually in cases where the automatic identification is not successful.	I
F5.2	Applying license for a module.	Relevant, e.g., in cases where the source files don't have any indication of the used license, but the module documentation clearly states the used license.	I
F6.1	Manual license template addition.	To improve the license identification coverage, the user is able to add new license identification template, e.g., in case of a new license.	I
F6.2	Run-time license template addition.	Used, e.g., in cases when license is indicated in the files on a previously unknown way.	I
F7.1	Visualization of license compatibility rules.	To show the user how license compatibility's are interpreted and analyzed.	A
F7.2	Definition of license compatibility rules.	Allows users to define compatibility rules according to their interpretations or standards.	A
F8.1	Visualization of the dependency map.	Visualizes the program parts and their dependencies to identify potentially reusable components and licensing problems.	A
F8.2	Retrieval of the details of each object in the dependency map.	Shows details of each dependency object: full file name, file type, license of the file, license status, list of licenses found from child objects, and (in the case of a source file) the actual source code.	A,S
F8.3	Browsing of the dependency map.	User is able to see the list of all the analyzed objects and their dependencies.	A
F9.1	Statistical and summary information about the licenses. found and the files used from the source package	Helps in satisfying the license obligations by providing summary information about the licenses found and the files used from the source package.	S

* Refers to step of the license compliance process: identify (I), approve (A), and satisfy (S).

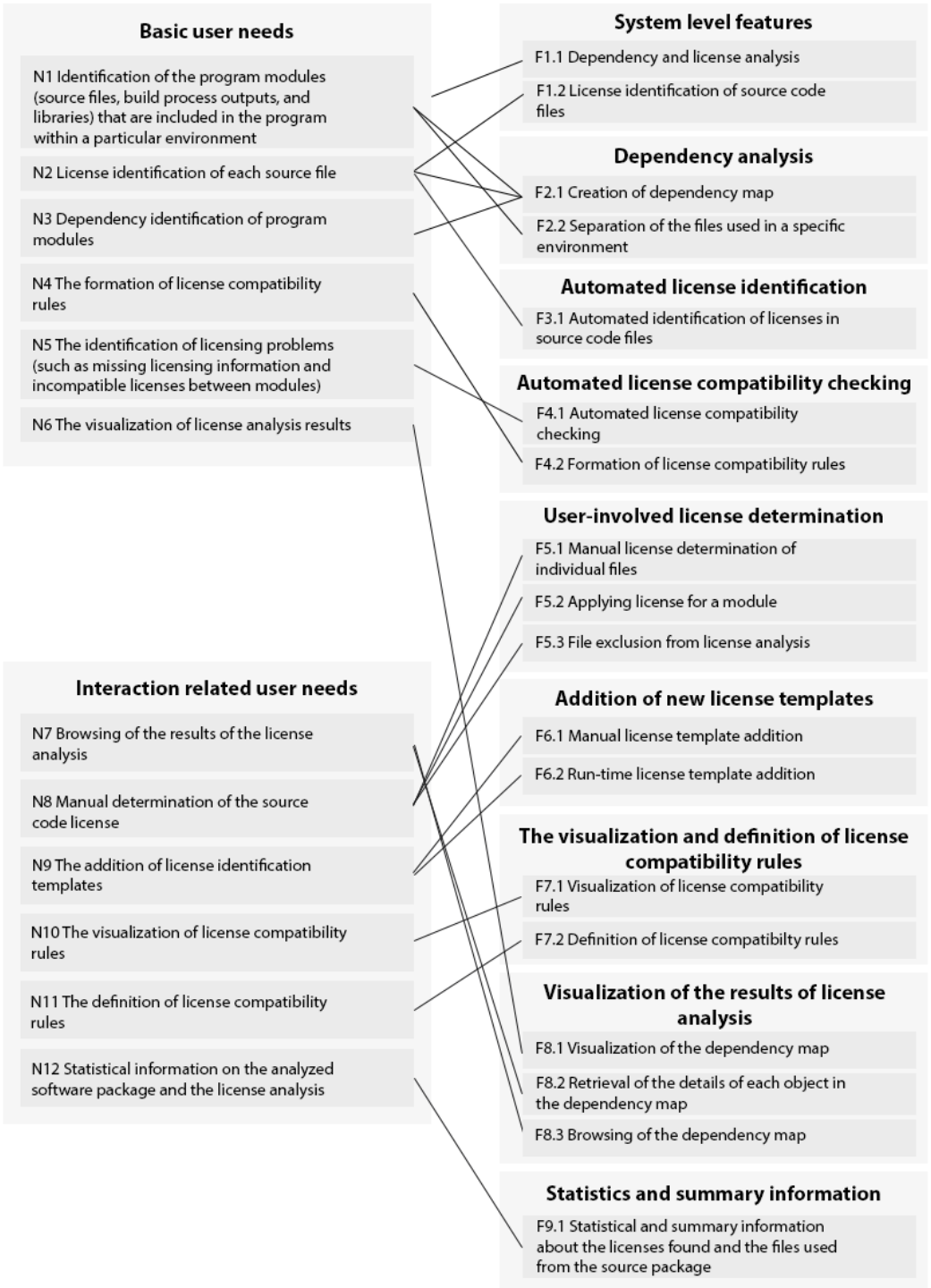

FIGURE 5 User needs and features of ASLA (Tuunanen et al. 2009).

4.1 Identification of licenses and their dependencies

In the identification step of the license compliance process, the reuser of OSS must identify the reused component or the snippets and their license(s). To get a detailed list of used source files, it also relevant to identify which parts are actually reused (N1) as large portions of software may not used at all (e.g., hardware architecture-specific portions of Linux kernel). In the identification step, the licenses of each individual source code file need to be identified (N2) to make a comprehensive analysis, since a single licensing bug, such as incompatibly licensed file, can prevent the reuse of the whole software package. In addition, the dependencies between these source files and the other objects (N3) (compiled or fetched modules) must be identified to later conduct the license compatibility analysis. Also, this identification can give some clues about reusable components within a larger software package, and this becomes useful when partial reuse is considered.

The most typical usage of the tool is to conduct a dependency and license analysis (F1.1). The usage process of this system-level feature is summarized in Figure 6. This full functionality is achieved by analyzing the OSS implemented using programming languages that can be compiled using GCC. ASLA can also identify the licenses without a dependency analysis (F1.2) (see Figure 7), of software packages that cannot be compiled using GCC, such as, for example, Perl, Java, or even HTML. These two features form the system-level features of ASLA. The system was tested using GCC version 4.1.3., and it also used slightly instrumented versions of ar (an archive tool) and ld (a linker) from GNU Binutils (version 2.18.50) (Tuunanen et al. 2009).

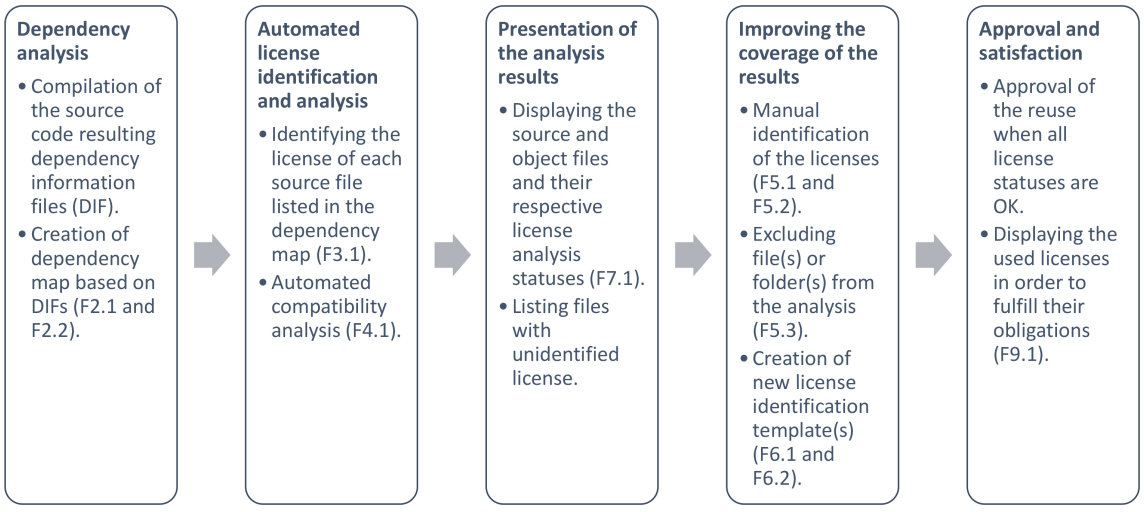

FIGURE 6 Dependency and license analysis (F1.1) usage process of ASLA.

For a dependency and license analysis (F1.1), we first identify all objects (source files, compiled objects, libraries, etc.) and dependencies between these objects (Tuunanen et al. 2006a). A dependency analysis forms the basis for the

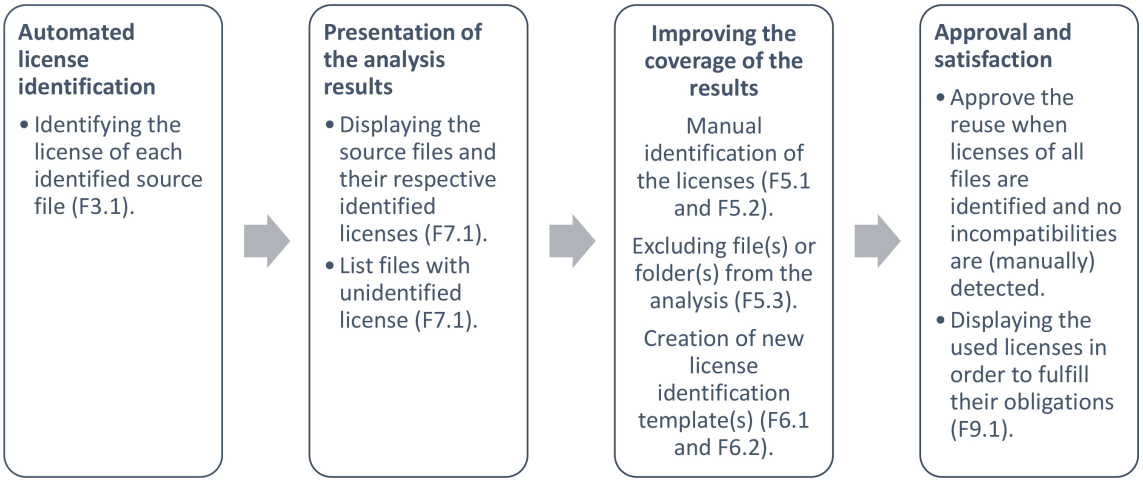

FIGURE 7 License identification (F1.2) usage process of ASLA.

advanced compatibility analysis between identified licenses. ASLA uses the outputs produced by GCC, GNU ar, and GNU ld. These programs, which are used in the build process of the software, store information on the dependencies between program parts to dependency information files (F2.1). ASLA reads these files, creates a map of dependencies, and performs the actual license analysis for the source files listed in the map (Tuunanen et al. 2006a). In this way, we get detailed dependency information and can leave out the source files not used in the selected environment (F2.2). Each object in the dependency information map has references to the objects that it is dependent on, and references to the objects that are dependent on it. At the time of the implementation, ASLA was known to be the only license analyzer that provides full information on build process outputs and their dependencies (Tuunanen et al. 2009). Each compiled object gets its license information as a composition of its source files' licenses. During the dependency map creation, a special treatment of .lo files (created by GNU Libtool), symbolic links, and duplicate files was required to ensure the correctness of the dependency map. The approach is simple, and it performs well even with large software packages, but it has one drawback. The overall performance of the license analysis is affected, as the analyzed package needs to be compiled.

Our goal for the license identification was to automatically identify the licenses of all the source files. In most cases, the OSS source files include either the full license text or predefined template or link to the license indicating the use of a specific license. Simple permissive open source licenses, such as BSD and MIT, are typically included at the beginning of each source code file as a whole. Another common practice is to make a reference to the license from the source code. For this purpose, predefined templates are often used. This technique is typical for open source licenses such as Apache, GPL, LGPL, and MPL that have longer license text. License identification is based on reading these license statements from each source code file. Automated license identification in ASLA was achieved by using the license templates given as regular expressions (F3.1). Regular expressions were chosen, because exact matching techniques were not feasible for several reasons: (i) comment characters and the various kinds of

white space characters prevent exact matching, (ii) programmers modify the predefined license texts, and (iii) there are different published versions of the licenses (Tuunanen et al. 2006a,b).

Before the search of the license text was conducted for a particular source file, there were some cleanups done for the source file, such as removing the comment characters (Tuunanen et al. 2006a). For example, the license search template (regular expression) for LGPL version 2 and 2.1 was as follows (Tuunanen et al. 2009):

is free software; you can redistribute it and/or modify (it)? under the terms of the GNU (Library)—(Lesser) General Public License as published by the Free Software Foundation; either version 2., or \(at your option\) any later version .* 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 (Library)—(Lesser) General Public License for more details You should have received a copy of the GNU (Library)—(Lesser) General Public License along with this (program)—(library); if not, write to the Free Software Foundation .* .*USA*

The actual identification in ASLA is performed using Java *java.util.regex.Matcher* and *java.util.regex.Pattern* classes. Our regular expression is applied as a multiline pattern for a source file that has been stripped for comment characters as follows (Tuunanen et al. 2009):

```
Pattern pattern = Pattern.compile(regexp, Pattern.MULTILINE);
Matcher matcher = pattern.matcher(sourceCodeString);
matcher.find();
```

Since the license of a source file is not always indicated in a previously known way, it must be possible to add new identification criteria for licenses (N9). New license identification templates can be added in two ways. The first way is to manually create a new text file in the directory in which the existing license template files are saved (F6.1). The file format for the new template contains the license name on the first line of the file and the template text in regular expression form in the following lines (Tuunanen et al. 2006a). ASLA automatically reads these new files in the next startup. Also, it is common that the license is indicated in the files in a previously unknown manner. For this purpose, ALSA provides a run-time license template addition (F6.2) that is conducted after the initial analysis has been done and the results have already been presented to the user. The user is able to select a text in a source file, define a license name for this text, and use that information as a license identification template, as shown in Figure 8 (Tuunanen et al. 2009). In this case, ASLA reformats the text into a regular expression and saves it for future use, if necessary.

In some cases, it is also convenient to determine the licenses manually for a single file or for a whole component (N8), as automated license identification cannot be achieved for every file. This is due to the fact that all the source files do not either indicate the used license or are indicated in a way that is not known

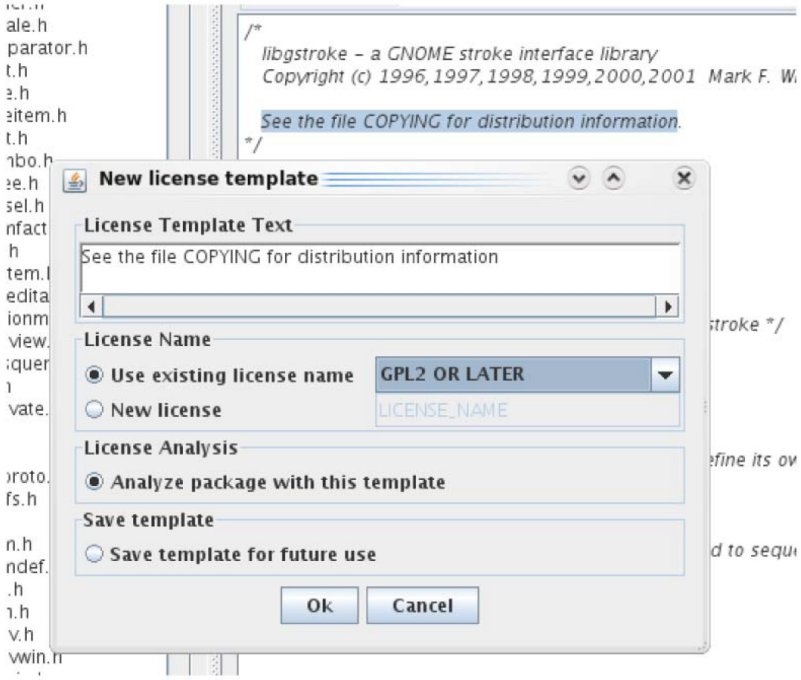

FIGURE 8 New license identification template addition in ASLA (Tuunanen et al. 2009).

before. There are two ways to manually identify the licence of a file, the first being to manually set licenses one by one for unidentified source files (F5.1). This is aided by the fact that ASLA lists all source code files that were unidentified in a separated tree entry (see Figure 9 (Tuunanen et al. 2009)). Another way is to apply a license for a whole module at once (F5.2). This is relevant in cases where the source files do not have any indication of the used license but where the module documentation (e.g., file called COPYING or LICENSE) clearly states the used license.

A problem sometimes faced in the license analysis is the inclusion of a library header file that has no license information whatsoever. These files cause ASLA to report on the missing license, even though such libraries are usually under permissive licenses that allow them to be used with very little restrictions as part of another program. For these types, ASLA provides the functionality to exclude a single file or a whole directory from the license analysis (F5.3).

By using the dependency and license analysis features described above, the tool fulfills the needs of the identification step of the license compliance process. By using these methods, we can obtain complete information about the used licenses and their relations.

4.2 Approval of OSS and satisfaction of license obligations

The identification of licenses and dependencies between files forms the base for the approval step of the license compliance process. If more than one license is found in the identification step, it must be evaluated whether this results in li-

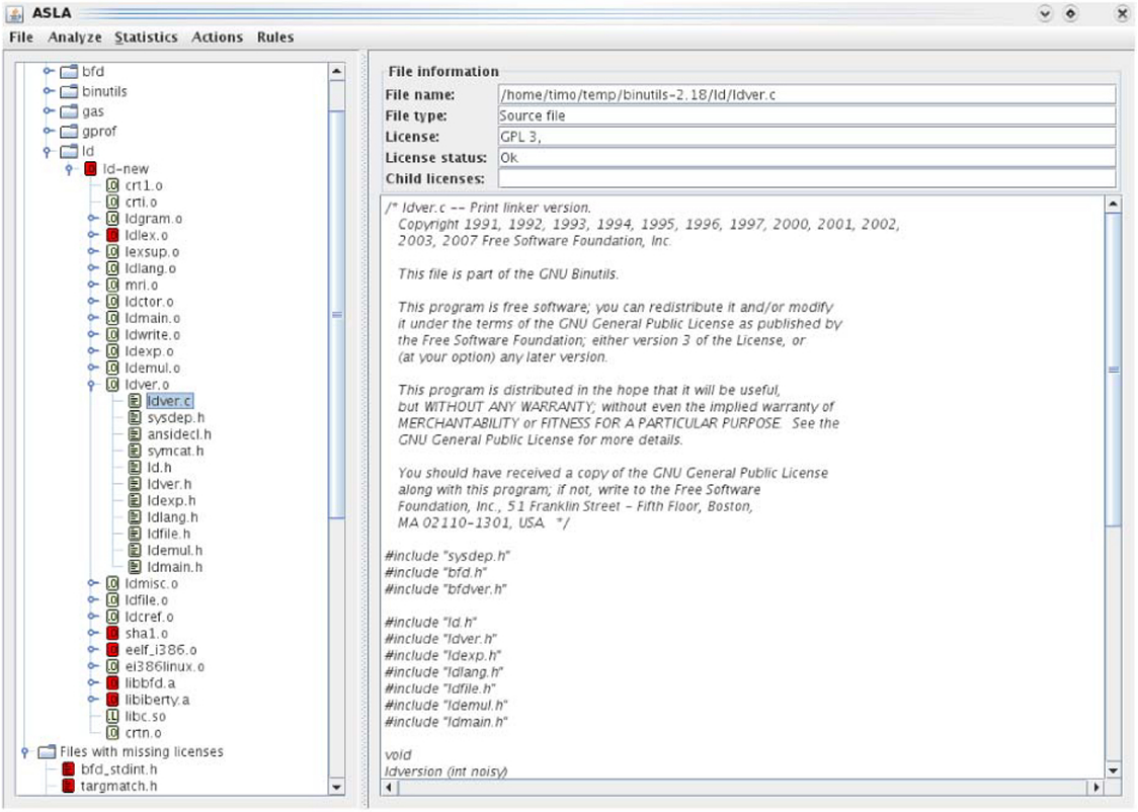

FIGURE 9 ASLA main view (Tuunanen et al. 2009).

cense incompatibility. An automated system needs rules that define the compatibility between the licenses (N4) to automatically detect incompatibilities. This automated detection of potential licensing problems (N5) is a fundamental feature of the license compliance tool. This need is dependent on the first four needs (i.e., N1–N4), and it cannot be fully satisfied unless the information from the first four needs is available.

To achieve extendable automated compatibility checking (F4.1), ASLA includes compatibility rules between licenses (F4.2). A license compatibility rule defines how two licenses cooperate with each other. There are four possible states that can be defined (Tuunanen et al. 2009):

1. NOK (Not OK). Used in situations of clear incompatibility (e.g., Apache 2.0 and GPLv2).
2. Warning. Used in situations where the compatibility is either unclear or where the reuser is willing to receive notification (e.g., use of an original BSD license with an advertising clause).
3. OK. Used when licenses are compatible with each other.
4. N/A (not available). Means that no rule operating between the two licenses has been defined.

License rules were defined for both dynamic and static linking, because licenses may behave differently depending on the linking style. Also, the relation between two licenses was defined twice, depending on the dependency direction (e.g.,

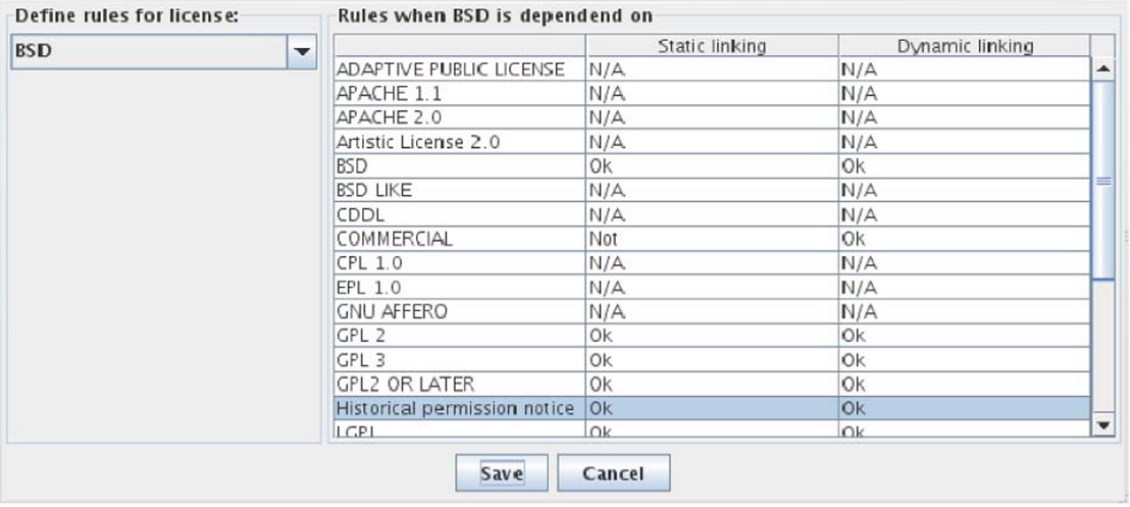

FIGURE 10 ASLA license compatibility rules view (Tuunanen et al. 2009).

MIT code is dependent on GPL licensed code or vice versa), making four different combinations between each of the two licenses.

As the compatibility of the OSS licenses can be interpreted differently (e.g., in case of specific community standards), the actual interpretations of the license compatibility rules must be presented (N10) to the user, and then, it is left to the user to define (N11) the rules between the different licenses. To fulfill these needs, ASLA offers a user interface (see Figure 10 (Tuunanen et al. 2009)) that is used for the visualization (F7.1) and definition (F7.2) of compatibility rules.

When the license analysis is conducted, the results need to be presented to the user (N6) in a way that allows for the user to make an educated decision of the acceptance of the reused software. The user must have the possibility to browse the individual files, their licenses, their dependencies, and the list of licenses found in the package (N7). These needs are also relevant in the satisfy step of the license compliance process (see Chapter 3). The ASLA user interface in Figure 9 (Tuunanen et al. 2009) has two main sections: the left side of the user interface displays the dependency map in tree format (F8.1), while the right hand side displays detailed information about the selected dependency object (F8.2). By browsing the tree on the left (F8.3), the user can see the list of all the analyzed objects and their dependencies.

The program parts that are successfully analyzed from the license perspective are colored green, and the objects that have some sort of potential license problem are colored red. An identified license problem includes an unrecognized license or the use of incompatible licenses. By selecting an object in the tree, the user is provided with the following detailed information concerning the object in question: *full file name*, *file type*, *license of the file*, *license status* (OK, unrecognized license, unrecognized child licenses, incompatible licenses), *list of licenses found from child objects*, and (in the case of a source file) the *actual source code*. The license status displays the information concerning the results of the license analysis, as listed in Table 4.

It is also beneficial that the statistical information of the analysis process

TABLE 4 ASLA license analysis statuses.

Status	Explanation
OK	License of the file and the licenses of all its child objects have been successfully identified and are identified to be compatible with each other.
Unrecognized license	License of the file has not been successfully identified.
Unrecognized child licenses	License(s) of some of the child files have not been successfully identified.
Incompatible licenses	All the licenses have been successfully identified, but licenses are incompatible with each other, as defined by the compatibility rules.

be summarized for the user (N12) because it helps in satisfying the license obligations. During the license analysis, ASLA collects statistical information and provides statistical and summary information about the licenses found and the files used from the source package (F9.1). ASLA lists all the licenses applied, the number of files with each applied license, and the number of files whose licenses were not recognized. The statistics also include information about the number of files from the source package and the number of external files included in the final binary output.

The ASLA features presented above provide the reuser of an OSS package automated means to identify the licenses and potential licensing problems. This aids the user in the approval or disapproval of the selected package or component. It also lists the used licenses assisting in fulfilling the licensing obligations.

4.3 Tool evaluation

In the development of ASLA, the main goal was to improve the efficiency of the analysis versus existing tools or compared with a manual analysis (N13). As described in Chapter 3, some open source licenses are incompatible with each other. Since ASLA is able to identify the licenses of source files and the dependencies between build process outputs, we were able to automatically identify the possible license compatibility problems in software components using the compatibility rules. This automated compatibility analysis greatly increases the efficacy of OSS license compliance process, as OSS packages can contain hundreds or thousands of source files with different licenses.

The license identification precision and coverage had the aim of being as high as possible (N14). However, this can vary significantly, since many packages contain licensing bugs such as missing licenses. Our goal was to identify the license of each file that can possibly be identified using both automated and interactive techniques. We evaluated our approach first by running a license analysis for 12 OSS packages of different sizes (Tuunanen et al. 2009). A summary of these results are presented in Table 5. In the case of dependency and license analysis (F1.1), which comprised nine out of 12 packages, we read and analyzed from 10 to 2532 dependency information files per open source package. We were initially able to identify licenses in 1–97% of the source files listed in those files without any user involvement (FI). The results were improved to identification of 75–100% of the files by applying new license templates (F6.2) (FI_F), as shown

TABLE 5 ASLA evaluation summary (Tuunanen et al. 2009).

Package	KLOC	FI	FI_F	Time
AFPL Ghostscript	673	1%	96%	67 s
Apache HTTP Server	316	5%	98%	13 s
Azureus *	25	56%	84%	20 s
Bugzilla *	162	75%	98%	1 s
GIMP—The GNU Image Manipulation Program	863	91%	92%	325 s
GNU Binutils	1289	88%	92%	18 s
GNU Go	134	35%	92%	124 s
gnuplot	736	41%	85%	11 s
JBoss *	177	97%	97%	18 s
Mozilla Firefox	2063	96%	98%	272 s
Pidgin IM client	332	86%	90%	216 s
Subversion	627	12%	100%	21 s

KLOC = Size of the source package in thousands of lines of code
FI = Percentage of source files identified by ASLA without any additional user involvement
FI_F = Percentage of source files with an identified license after user involvement
* Indicates license identification (F1.2). Other packages include dependency and license analysis (F1.1).

in Figure 8. Most of the instances of non-identification were caused because the source files had no indication whatsoever of the license used. In the case of license identification (F1.2), we were also able to initially identify licenses in 1–97% (FI) of the source files. By applying new license identification templates (F6.2) and by excluding unnecessary files (F5.3), such as HTML help files, the identification results improved to between 79% and 98% (FI_F).

The efficiency of the license identification of source code files (F1.2) was between one and 472 files per second. This large variance can mostly be explained by two factors: (1) Successful identification is usually very efficient (some milliseconds/file), especially when most files within the package have the same license. (2) In the case of slow results, the license identification was unsuccessful for most files. The dependency analysis (F2.1 and F2.2) and compatibility checking (F4.1) also affect the total efficiency. In most cases, the overhead produced by these features was about 1.5 to 2 times the time spent on license identification, but in some cases, this figure rose as high as eight-fold.

The evaluation provided two main results: (1) the ASLA tool enables the analysis of large OSS programs in a reasonable time, and (2) the tool provides information which is valuable for meeting license compliance when reusing OSS. The tool is easily extendable, and it is not restricted to any particular programming language.

4.4 Validity

As ASLA was created according to the DSR paradigm, the artifact construction can be validated accordingly. Sonnenberg and Vom Brocke (2012) identify four evaluation types derived from typical DSR activities. These types relate to DSR processes that include the activities of problem identification, design, construction, and use. Evaluation activities include the following (Sonnenberg and Vom Brocke 2012; Vom Brocke et al. 2020):

Eval1: The evaluation of the problem identification activity serves the purpose of ensuring that a meaningful DSR problem is selected and formulated.
Eval2: The evaluation of the solution design serves the purpose of showing that an artifact design progresses to a solution of the stated problem.
Eval3: The evaluating of the solution instantiation serves to demonstrate if and how well the artifact performs.
Eval4: The evaluating of the solution in usage serves to ultimately show that an artifact is both applicable and useful in practice.

As our evaluation occurs after the instantiation of our artifacts, it is an ex post evaluation (Venable et al. 2016). The information in Table 6 displays the results of this post-evaluation.

TABLE 6 Evaluation of ASLA to DSR checklist according to Vom Brocke et al. (2020).

Phase*	DSR question	ASLA
Eval1	Is the importance of the research stated in a justified manner?	There was very little prior research available related to automated license analysis at the time of the writing of articles related to ASLA. Some tools such as OSLC and FOSSology existed, but those were not described academically. Thus, the need for the automated license analysis was explicitly stated based on real-world problems, especially in the form of user needs (Tuunanen et al. 2009).
	Is the novelty (research gap) of the approach clearly stated?	Shortcomings of the other tools existing at the time were analyzed and listed in Tuunanen et al. (2009). Especially dependency analysis and automated compatibility checking were novel solutions. Also, improved license identification was needed.
	Is the feasibility (design objectives) of the approach stated in justified manner?	User needs formed the objectives for our design. Our main objective was stated in the need 13: goal when developing ASLA was to improve the efficiency of the analysis versus existing tools or compared to manual analysis.
Eval2	Does the design specification meet the requirements of simplicity, clarity, and consistency?	Features (see Table 3) and needs (see Table 2) are clearly listed and linked to each other, as displayed in Figure 5. Also, the tool architecture was carefully designed, as presented in Figure 4.

	Is the selection of design methodology justified?	Because of its overall novelty, we used a prototype-based iterative and incremental development process. The benefits of these development process models were well known at the time of the development.
Eval3	Is the validated artifact easy to use?	All features were based on real-world needs, and the relevant information is provided in the graphical user interface (see Figure 4). The user interface highlights the files with successfully identified licenses and potentially problematic parts with separately colored nodes. Also, the identified licenses are collected as a separate node. By this way, the user is able to quickly find the relevant information. Full functionality that includes a dependency analysis is achieved using well-known compilation procedures.
	Does the validated artifact perform its task of solving a real-world problem?	ASLA implements all identified user needs for automated license analysis that were reported in Tuunanen et al. (2009).
	Is the validated artifact robust in its tasks?	ASLA's validation was reported in Tuunanen et al. (2009). The results of this evaluation are summarized in Table 5. This evaluation did not reveal any shortcomings in the robustness of the dependency and license analysis.
Eval4	Is the validated artifact effective, efficient, and externally consistent in its tasks?	As stated in the evaluation, ASLA enables the analysis of large OSS programs in a reasonable time, and it provides information which is valuable for meeting license compliance when reusing OSS.

* Refers to the evaluation phase

4.5 Summary

In this chapter, we have described our reverse engineering approach for automated license compliance and its implementation, ASLA. Our implementation supports all 12 identified user needs that are listed in Table 2. The evaluation of ASLA revealed that it enables the analysis of large OSS programs in a reasonable time and provides valuable information for meeting licence compliance. Next, we will present the results from the review cycle.

5 RESULTS FROM THE REVIEW CYCLE

Systematic literature reviews and mapping studies have increased their popularity to better understand the empirical basis of software and systems development (see, e.g., Al-Zubidy (2017) and Banaeianjahromi and Smolander (2016)). This chapter presents the review cycle in the form of a systematic literature review that describes how automated OSS license compliance has evolved during the 2010s. The following research questions are answered for the review cycle part:

RQ1 What are the user needs to fulfill automated open source license compliance?
RQ2 What software features are needed to fulfill the user needs?

As we are answering these questions based on the state of available public information, we are dealing with a secondary study. As is customary, this secondary study follows a systematic approach. As Kitchenham et al. (2015) state, systematic reviews are used to evaluate how far particular techniques have been adopted by industry and commerce or to identify the benefits of using tools in a particular context. As we are interested in offering a comprehensive view of the methods and tools of the automated license compliance process, a qualitative approach is being adopted. Qualitative reviews usually address questions about the specific use of technology, so they are unlikely to involve making comparisons (and hence less likely to address questions that involve any sense of something being "better") (Kitchenham et al. 2015).

5.1 Research protocol

This section will describe the details of the methods used in the SLR. It includes the searching for and inclusion of the studies, as well as the data extraction and synthesis methods.

TABLE 7 Summary of the automatic search.

Engine	Search expression	Years	Hits*	Yield†
Google Scholar	(intitle:licensing OR intitle:licensed OR intitle:licenses) "open source" tool OR method OR approach	2009 – 2020	604	78
Google Scholar	(intitle:license -plate) "open source" tool OR method OR approach	2009 – 2020	375	86
Google Scholar	(intitle:violation OR intitle:violations OR intitle:inconsistency OR intitle:inconsistencies OR intitle:compliance) "open source" clone detection license	2009 – 2020	59	22
Google Scholar	(intitle:compliance) "open source" license	2009 – 2020	424	38
Google Scholar	intitle:"open source" intitle:legality "open source" license	2009 – 2020	7	4
Web of science	AB=("open source" AND license NOT plate) AND AB=(violation OR violations OR inconsistency OR inconsistencies OR compliance) AND SU=Computer Science	2009 – 2020	11	9
Web of science	TI=(licens* NOT plate) AND AB=("open source")) AND SU=Computer Science	2009 – 2020	16	14
Web of science	AB=(licens* NOT plate) AND AB=("open source")) AND SU=Computer Science	2009 – 2020	446	23
Web of science	(TI=compliance AND AB=("open source" AND license) AND SU=Computer Science	2009 – 2020	2	2
IEEE Xplore	("Abstract": "open source" AND "Abstract": "license") AND ("Abstract": "violation" OR "Abstract": "violations" OR "Abstract": "inconsistency" OR "Abstract": "inconsistencies" OR "Abstract": "compliance")	2009 – 2020	16	13
IEEE Xplore	("Abstract": "open source" AND "Abstract": "license") AND ("Abstract": "tool" OR "Abstract": "method" OR "Abstract": "approach")	2009 – 2020	108	16
ACM dl	[Abstract: "open source"] AND [Abstract: "license"] AND [[Abstract: "violation"] OR [Abstract: "violations"] OR [Abstract: "inconsistency"] OR [Abstract: "inconsistencies"] OR [Abstract: "compliance"]] AND [Publication Date: (01/01/2009 TO *)]	2009 – 2020	11	11
ACM dl	[Abstract: "open source"] AND [Abstract: "license"] AND [[Full Text: "method"] OR [Full Text: "tool"] OR [Full Text: "approach"]] AND [Publication Date: (01/01/2009 TO *)	2009 – 2020	46	9
Springer	"open source" AND (violation OR violations OR inconsistency OR inconsistencies OR compliance) AND NOT (plate)	2009 – 2020	29	13
Springer	license AND "open source" AND (method OR tool OR approach) AND NOT (plate)	2009 – 2020	35	12

* Hits refers to the number of search results obtained, as reported by the search engine.
† Yield refers to the number of candidate publications recorded (may include some of the same candidates as other searches).

5.1.1 Searching for candidate studies

The search for the candidate studies was done in two steps: first, we made an automated search, and second, we conducted a snowball search. The search phrases for automated searches were formulated carefully to include relevant publications and minimize irrelevant candidates. The data collection took place in September 2020 and encompassed five databases: Google Scholar, Web of Science, IEEE Xplore Digital library, ACM digital library, and SpringerLink. A summary of the searches and their results is shown in Table 7.

Our initial goal was to make a multi-vocal literature review that would also include results from "gray" literature, such as blogs, white papers, and web pages, such as, for example, in Garousi and Mäntylä (2016). As we ran searches on Google for the following search terms: *"open source" license compliance tool* and *"open source" license tool*, we got 43,000,000 and 121,000,000 hits, respectively. These numbers are not exceptional, and our goal was to utilize the relevance ranking of the search engines (e.g., Google's PageRank algorithm) to restrict the search space. This means that the most relevant results are displayed in the first few pages and that we would not have to go through more than 10 or 15 first pages of the results. However, after browsing through more than 20 pages of the first searches results, there was no sign of relevance saturation. For exam-

ple, result 143 was a blog post describing Licensed, a GitHub tool for maintain dependency license documentation (Ruskin 2018), which appeared relevant. Further down the list, result 259, Gangadharan et al. (2012), was already found in the automated search done using scientific engines. Also, the number of relevant sources was very low, limited mostly to lists of available tools without detailed information about their features or comparison of the tools. This lead to a conclusion that it is not possible to make a comprehensive inclusion of the "gray" literature, so this approach was dropped, meaning that only academic literature was included.

For the validation of our searches, we selected four papers that should be found by the searches. All of these papers use ASLA (Tuunanen et al. 2009) as their reference. These papers include German et al. (2010b), Kapitsaki et al. (2015), Paschalides and Kapitsaki (2016), and Wu et al. (2015). All these articles were found in automatic searches.

The second step, a snowball search, was made on the relevant sources found in the automated searches. First, we manually scanned the reference list of each article. Then, we searched the article in Google Scholar and scanned the referencing articles as well.

5.1.2 Selection of studies

We carefully defined the inclusion and exclusion criteria to ensure all relevant sources were included in the study and all out-of-scope sources were left out. The inclusion criteria are based on the three-step license compliance process of OSS reuse: *(i) identify* used OSS, its origin, licenses, and dependencies, *(ii) approve*: review the output from the previous step, understand the licenses that govern the use, modification and distribution of the source code in question, and make a decision on the approval or disapproval of the use of the identified OSS, and *(iii) satisfy* the license requirements: license, copyright, and attribution notices for all approved OSS (whole components and snippets) are included in the product documentation. Thus, the inclusion and exclusion criteria are as follows:

1. Is this a primary study that describes an automated feature(s) or a method(s) to create an (at least partially) OSS bill of materials for reuse? The bill of material identifies all OSS used (packages and snippets), their origin, license, and any licensing mismatches. Also, it includes listing other OSS licensing bugs in the source code and other potential intellectual property violations such as illegal copying.
2. Is this a primary study that describes an automated feature(s) or method(s) assisting in OSS reuse approval? Methods or tools include ways of helping in license understanding or help in deciding on the approval or disapproval of OSS reuse from the licensing perspective.
3. Is this a primary study that describes an automated feature(s) or method(s) assisting in satisfying the open source license terms when reusing OSS? For instance, is this done by listing the license, copyright, and attribution notices for all approved OSS.

4. Is this a secondary study that describes the feature(s) or methods(s) that answer one or more of the questions above?
5. Is the study related to a reusable form of software (source code or Java Archive (JAR)) which includes the licensing information?
6. Is this study peer reviewed?
7. Is this study written in English?
8. Is this study published after 2009?
9. Does the research appear sound?

The first four questions are the inclusion criteria, and the final five are the exclusion criteria. A source was excluded if an answer to all of the first four questions or any of the remaining was negative. To validate these criteria, a set of 18 studies were provided to the supervisors of this dissertation for validation of the inclusion/exclusion criteria. This set included studies that the author had both included and excluded based on the criteria. Only two of these studies raised a debate whether they should be included or not. The rationale for the borderline cases are described below.

Sources that did not meet the above criteria were excluded. For example, sources that simply use the license analyzers to gather data were left out (e.g., Vendome et al. (2017a); Manabe et al. (2010)). Sources that simply identify code clones but are not related to license identification or IPR violation detection from a reuse perspective were left out (e.g., Feng et al. (2019), Hemel et al. (2011), and Sajnani et al. (2016)). Also, papers that try to identify licenses from binary files such as Duan et al. (2017) were excluded. Papers describing the methods for simply selecting licenses were excluded (e.g., Viseur (2016)), unless they present some sort of a method for license comprehension (e.g., Kapitsaki and Charalambous (2016, 2019)), thus being part of the approval step of the license compliance process. In addition, even though, for example, the titles *A study on the identification of open source license compatibility violations* (Lee and Seo 2018) and *Method for License Compliance of Open Source Software* (Yun et al. 2017a), seemed very relevant, but they were not written in English and, so they were excluded. Some studies raised questions on the quality of the research as they were only few pages long (e.g., Gordon (2014), Heirendt et al. (2017), and Xu et al. (2010)). These were excluded as well. One paper, Singi et al. (2019), which could have been excluded as it is more of an framework than actual tool or method for license compliance, was included, however. The reason for this is that it offers a framework that could be used for implementing an actual license analysis and compliance assisting software.

Every source located during the searches were subject to the three phase selection procedure, summarized in Figure 11. Outcome of each phase was either exclusion or inclusion. The first phase was conducted during the searches: title, abstract, keywords, and other available metadata were examined. Only sources that were obviously outside the scope were excluded, and all other sources were included. The second phase was done based on the same metadata as phase one, and the full text was briefly examined if the metadata were not very useful. In the

second phase, we recorded all exclusions with an explanation. The third phase was to read the full text of the remaining sources and keep a record of inclusion and exclusion subjects with an explanation. A list of articles that were excluded in the third phase of the selection procedure is found in Appendix 2.

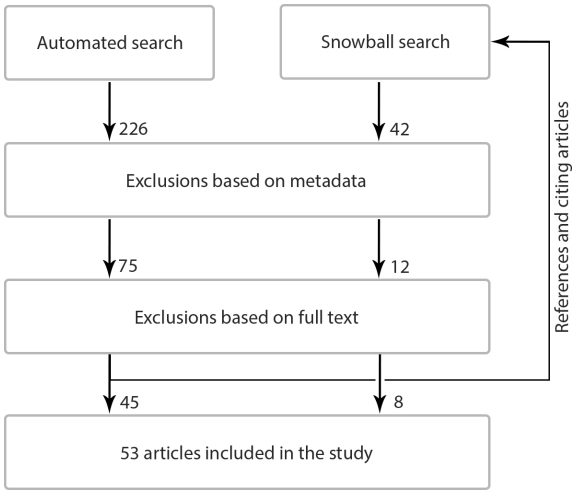

FIGURE 11 Flow diagram of the study selection procedure.

For the automated search, phase 1 resulted in 226 individual studies after the duplicates had been removed. After phase 2 had been conducted, 75 studies were left. Out of these 75 studies, 45 were included in the review. Snowball searching resulted in a further 42 studies for phase 1. After phase 2 was conducted, 12 of these entered phase 3, and eight were included. Thus, the total count for the included studies is 53, as presented in Table 8.

TABLE 8 Included publications.

Study	P/S*	Authors	Type†	Forum‡	Steps○
S1	P	Alspaugh et al. (2010)	M	J	A
S2	P	Alspaugh et al. (2011)	M	C	A
S3	P	Alspaugh et al. (2012)	M	C	A
S4	P	Alspaugh et al. (2013a)	M	Bc	A
S5	P	Alspaugh et al. (2013b)	M	Bc	A
S6	P	Bavota et al. (2014)	F	C	A
S7	P	Bei and Yuan (2013)	M	C	I
S8	P	Davies et al. (2011)	F	C	I
S9	P	Davies et al. (2013)	F	J	I
S10	P	Di Penta et al. (2010a)	M	C	I
S11	P	Di Penta et al. (2010b)	M	C	I
S12	P	Dyck et al. (2016)	M	C	I,A,S
S13	P	Dyck et al. (2018)	F	C	I,A,S
S14	P	Eghan et al. (2019)	M	J	A
S15	P	Gangadharan et al. (2012)	F	J	A
S16	P	German et al. (2010a)	M	C	A
S17	P	German et al. (2010b)	F	C	I
S18	P	German and Di Penta (2012)	M	J	I,A
S19	P	Golubev et al. (2020)	M	C	I
S20	P	Gordon (2011)	F	C	A

S21	S	Hemel (2015)	N/A	J	I,A,S
S22	P	Higashi et al. (2016)	F	C	I
S23	P	Higashi et al. (2019)	F	J	I
S24	P	Inoue et al. (2012)	F	C	I
S25	P	Ishio et al. (2016)	F	C	I
S26	P	Jaeger et al. (2017)	F	J	I,A,S
S27	P	Kapitsaki and Kramer (2015)	F	J	A
S28	S	Kapitsaki et al. (2015)	N/A	J	I,A,S
S29	P	Kapitsaki and Charalambous (2016)	M	C	A
S30	P	Kapitsaki et al. (2017)	F	J	A
S31	P	Kapitsaki and Paschalides (2017)	F	C	A
S32	P	Kapitsaki and Charalambous (2019)	F	J	A
S33	P	Kashima et al. (2011)	M	C	I
S34	P	Kechagia et al. (2010)	F	C	I
S35	P	Lee et al. (2015)	M	J	I
S36	P	Liu et al. (2019)	F	C	I
S37	P	Lokhman et al. (2012)	F	Bc	A
S38	P	Manabe et al. (2014)	F	Bc	A
S39	P	Mathur et al. (2012)	M	C	I
S40	P	Mattmann et al. (2015)	F	C	A
S41	P	Mlouki et al. (2016)	M	C	I
S42	P	Moreau et al. (2019)	F	J	A
S43	P	Nejad et al. (2016)	F	C	A
S44	P	Paschalides and Kapitsaki (2016)	F	C	A
S45	P	Pellegrini et al. (2019)	F	C	A
S46	P	Ragkhitwetsagul and Krinke (2019)	F	J	I
S47	P	Singi et al. (2019)	M	C	I,A,S
S48	P	Van Der Burg et al. (2014)	F	C	I,A
S49	P	Vendome et al. (2017b)	F	C	I
S50	P	Wu et al. (2015)	M	C	I
S51	P	Xu et al. (2010)	M	C	I,A
S52	P	Yun et al. (2017b)	F	C	I
S53	P	Zhang et al. (2010)	F	C	I,A

* Indicates whether the study is a primary (P) or secondary (S) study
† Describes a method(s) (M) or feature(s) (F) for license compliance
‡ Published in journal (J), conference (C), or as book chapter (Bc)
○ Refers to the step of the license compliance process: identify (I), approve (A), and satisfy (S)

5.1.3 Data extraction and synthesis

For qualitative systematic reviews and mapping studies, data are often extracted in textual form or through the use of a set of classification schemes (Kitchenham et al. 2015). As our overall aim is to identify, evaluate, and synthesize research about automated open source license compliance, this forms the basis for the classification and extraction. Extracted data included the following:

1. Publication details
2. Review-specific data relating to
   (a) Type of paper (primary or secondary study)
   (b) Scope of study in terms of OSS license compliance (new automated features or other methods)
   (c) Main topics covered in terms of OSS license compliance

(d) Has the approach been validated? If the validation is done, how has the validation been performed?
(e) Summary of the main results

In addition to review-specific data, publication details including publication year and publication forum (journal, conference, book chapter) were collected for each included paper. We classified the publications using the following classification criteria:

- Is the study a primary or secondary study?
- Does the study describe new features (F) for automated license compliance tools, or does it present a method (M) that, for example, uses a combination of tools to extract results or describe some other results related to license compliance?
- Which step(s) of the license compliance process does the paper address?

A summary of the papers in terms of publication details and classification are listed in Table 8. Papers in the selected literature that describe new features related to the OSS license compliance process are summarized in Table 10. Papers that do not describe a new functionality are treated as related work.

5.2 Overview of the results

In this section, we present the overview of the review cycle's results. To answer our research questions, a classification of the included studies is first required as it assists in identifying existing research approaches and concrete techniques (Kitchenham et al. 2015). An overview of the results is presented based on this classification. Detailed results related to each step of license compliance process are presented in individual sections after the overview.

5.2.1 Classification of the included studies

Included papers are distributed fairly evenly between the different years of the second decade of the millennium, showing that the interest in the field of license compliance has remained fairly stable over the years. However, this does not reflect the fact that OSS reuse and the amount of OSS produced every year has increased substantially over the last 10 years (Dorner et al. 2020). The distribution of papers between publication years is presented in Figure 12.

Most of the included studies are related to one step of the license compliance process, mostly concentrating on the identification and approval steps. The studies are related to the 1) identification of the origin (IO), 2) license identification (LI), 3) dependency identification (DI), 4) license compatibility analysis (CA), 5) license comprehension (LC), 6) satisfying the license terms (SA), and 7) other studies (OT). The distribution of papers between these are presented in Figure 13.

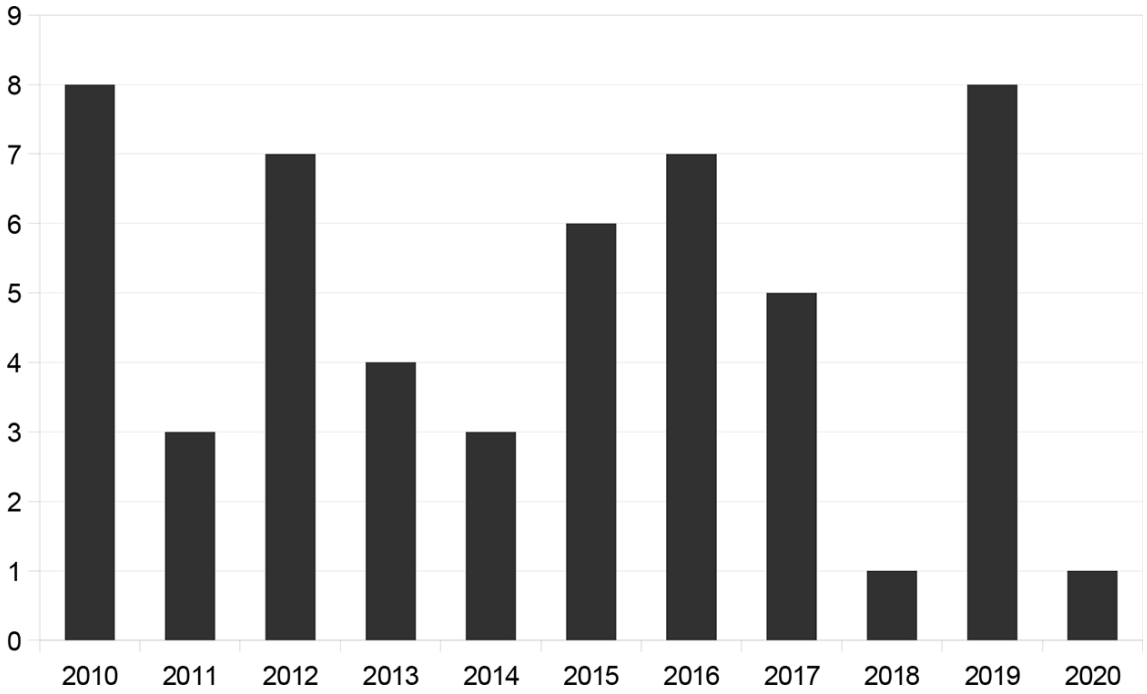

FIGURE 12 Distribution of papers by year of publication.

It should be noted that total number of studies in Figure 13 exceeds the number (53) of included studies, because some studies are related to more than one of the fields listed above. This distribution gives us a partial answer to research question RQ1.2 as it reveals fields of license compliance process that are identified to be relevant for users.

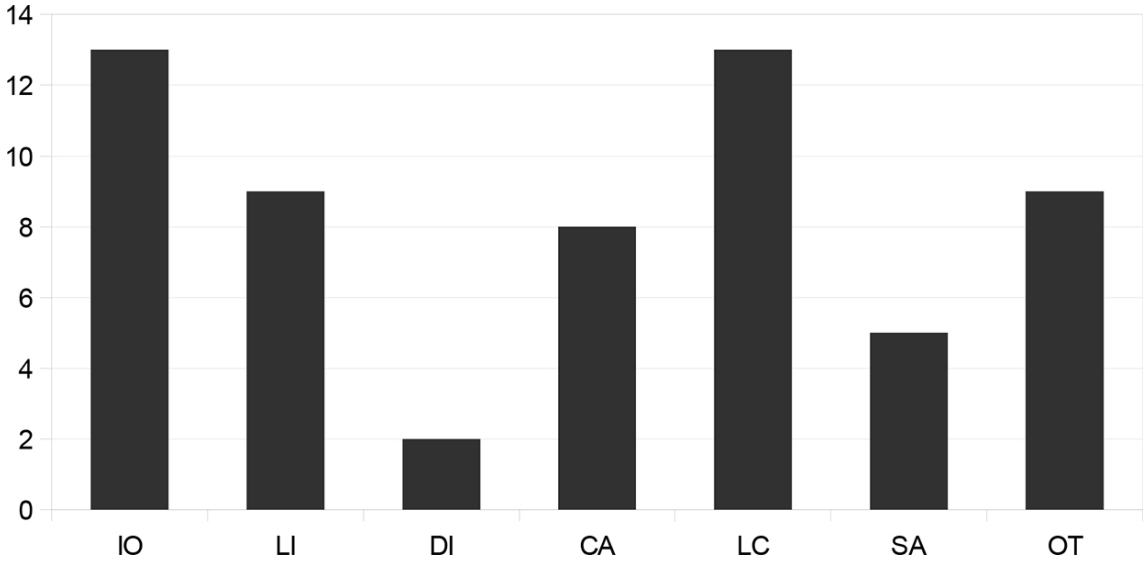

FIGURE 13 Distribution of papers between fields of license compliance.

Identifying the origin (IO) of the OSS typically involves using clone detection techniques in combination with other tools such as code search engines. Several license identification (LI) techniques are proposed in the selected literature. However, one of them, Ninka (German et al. 2010b) is considered a state-of-the-

art license analyzer (Mlouki et al. 2016). It uses a sentence-matching method to identify licenses from source files. The papers related to dependency identification (DI) include only a few different approaches. A license compatibility analysis (CA) and license comprehension (LC) have resulted in a wide variety of research, illustrating the complexity of these issues.

The papers describing tools that cover the whole license compliance process are limited to FOSSology (Jaeger et al. 2017) and Java-specific OMP (Dyck et al. 2018). These are also the only tools that address the satisfy step (SA) of the process beyond listing the identified licenses. The tools categorized to fields listed above (IO, LI, DI, CA, LC, and SA) form the group of primary studies of the SLR. The included studies also contain two secondary studies, that is, studies that are based on available public information: Kapitsaki et al. (2015) (S28) and Hemel (2015) (S21). Other studies (OT) include papers that were either vague in describing their results in respect to the license compliance process or lacked validation.

There is one thing that needs to be taken into account when the usage of online services is mentioned in the following sections. The services including Google Code, Google Code Search, and Koders that were used as part of research in the included literature are no longer available. Google Code was an online project hosting sites similar to GitHub, whereas Google Code Search and Koders were search engines used for finding OSS. To further increase the likelihood of confusion, at the time of the writing, Google offers a similarly named service called Code Search[1] that is used for searching exclusively Google's own open source code, such as code in Android or Chromium. This, however, is not referred to in any of the included literature.

5.2.2 Secondary studies

Kapitsaki et al. (2015) (S28) investigate approaches within three main categories: license information identification from source code and binaries, software metadata stored in code repositories, and license modeling and associated reasoning actions. The tools in each of these categories assist in OSS license compliance and help find potential licensing violations. The authors note that even though tools do not necessary solve potential violations, they assist in finding these violations faster. For license identification, they list many tools that are also covered in the current study such as ASLA, Ninka, FOSSology, and Joa, as well as some tools that were not described in our included studies. These license identification tools will be introduced in Subsection 5.3.2. The study (S28) includes also analysis of three code repositories in terms of how they assist developers in licensing issues: Sourceforge[2], Google Code, and GitHub. Kapitsaki et al. discover that even though these repositories allow the users to add licensing meta data into their projects, it is questionable whether this information for existing projects is available and reliable. However, these repositories can be used as an input to license

[1] https://developers.google.com/code-search accessed Jan 18, 2021

[2] https://sourceforge.net/ accessed Jan 18, 2021

analyzers as the source code is available in a suitable format. The authors also systematically cover the license meta-modeling and reasoners that we address in Subsection 5.4.2.

Hemel (2015) (S21) lists tools for open source license compliance in three categories: 1) license statement extraction, 2) copyright statement extraction, and 3) code clone detection. These tools focus on the license compliance of OSS in devices as reuse in these is extensive, market pressure for keeping prices is low, infringements are common, and software is distributed in a mix of source and binary code. The tools listed include FOSSology, Ninka, and CCFinderX. Hemel also lists the main topics that require more research, here being copyright extraction for satisfying the license obligations and determining who the real authors of software actually are.

5.2.3 Primary studies

The primary studies of the SLR total 44 articles. These are divided between papers that present automated features (20 papers) and papers describing other methods (24 papers) that assist in OSS license compliance.

As mentioned above, there were two tools that cover the whole license compliance process and were reported using scientifically sound methods: FOSSology and OMP. Their general information is presented here, and the details of their features in upcoming sections are divided between the steps of the license compliance process. The FOSSology project published the first version of its software in December 2007. Jaeger et al. (2017) (S26) describe the history and subsequent versions of FOSSology. It identifies the licenses from source code, lists license obligations, and provides documentation for satisfying license terms. Dyck et al. (2016) (S12) introduce an "organizational-technical concept for dealing with the various OSS licenses by using procedural instructions and build automation software." The technical aspects allow the use of build automation tools to support and verify open source license compliance during the software development life cycle. Implementation that supports the concept, called OMP, is described in another paper (Dyck et al. 2018) (S13). OMP uses the build automation tool Maven for data collection and the software repository tool Nexus for storing and fetching the collected information. The primary studies also describe tools that cover more than one step in the license compliance process but do not cover the whole process, such as Ninka and the unnamed tool described in Van Der Burg et al. (2014).

The only method described in the primary studies that covers more than one step of the license compliance process is Kenen (German and Di Penta 2012) (S18). Kenen is a semiautomatic process to help organizations in their open source license compliance for Java development. It involves Joa (Davies et al. 2011, 2013; Ishio et al. 2016) (S8, S9, and S25) for determining the origin of the JAR archive's source code, Ninka for license identification, and a manual licensing requirements analysis. German and Di Penta emphasise the need of license identification of each version of the component as the license may change between versions.

TABLE 9 User needs for automated open source license compliance identified in the review cycle.

Need	Step*
Identification of the origin of source files.	I
Identification of exceptions to known licenses.	I
Listing of rights, obligations, and restrictions of licenses found in analyzed package.	A,S
Proposal of alternate licenses that can be used for the package.	A
Reporting of the license analysis results in SPDX format.	A
Formation of acknowledgement documentation for each identified component including list of used licenses with full license text(s) and extracted copyright information.	S
Integration of automated license compliance features into development tools such as package managers or continuous integration tools.	I,A,S

* Refers to the step of the license compliance process: identification (I), approval (A), and satisfy (S).

For the license requirements analysis, they recommend a manual process that is performed by expert with software and legal knowledge. They justify this based on the fact that software and its dependencies are complex and that the licenses are written in legal terms and are intended to be interpreted by lawyers, not computers. German and Di Penta apply Kenen to Maven2 database that included more that 500,000 repositories with 275 Gbytes of archives to verify whether an example application uses open source and if it satisfies its licensing constraints. The preprocessing analysis of this repository created a signatures database and it took approximately 325 hours on a typical desktop computer. This highlights the complexity and computational requirements of clone detection on a large scale. Nevertheless, the actual identification of the clones of their example application took only a few seconds for each JAR file and successfully identified 27 of the 57 JAR files of their example application as existing in Maven2. Their analysis reveals that there were no incompatibilities between licenses, but acknowledgement requirements in the form of copyright notices were not fully satisfied.

All user needs that were not identified in the design cycle were collected from the primary and secondary studies. These user needs that address research question RQ1.2 and their relation to OSS license compliance process are summarized in Table 9.

All features described using scientifically sound methods were collected from the primary and secondary studies. These features that address research question RQ2.2 and their relation to OSS license compliance process are summarized in Table 10.

TABLE 10 Features for automated OSS license compliance identified in the review cycle.

Tool	Described In*	Used In	Identification	Approval	Satisfy
Ninka	S17 (S22, S23)	S16, S18, S26, S33, S38, S41, S48, S50	Sentence matching method to automatically identify licenses from source files.		Lists included licenses.
FOSSology	S26	S10, S27, S30, S35	Uses three different license identifiers and outputs results in SPDX format.	Allows for defining obligations and risks and associating them to licenses. Reports these on analyzed package.	Provides reports including licensing and copyright information.
OMP	S13		Resolves dependencies of the Maven project.	Downloads the license text and additional information about the license for each dependency. Checks rules that apply to license types. If the rules are not satisfied, the build process is aborted.	Source code and license texts are pulled from the Nexus repository and provided together with the products.
Joa	S8, S9 (S25)	S18	Uses anchored signature matching technique to identify the source origin for Java classes.		
Siamese	S46		Clone search on Java data sets, such as online code repositories. License identification based on pattern matching.		
Ichi Tracker	S24		Takes a code fragment as its query input, and returns a set of cloned code fragments which can be found by source code search engines.		
-	S48		Constructs dependency graph by tracing the operating system calls.	List deliverables whose source files are released under incompatible licenses.	
-	S49		Identifies license exceptions using machine learning.		

LChecker	S53	Utilizes Google Code Search service to check whether the analyzed file exists in an OSS project.	
SPDX-VT	S27, S30, S44		Examines the structure of SPDX for correct licenses and license compatibility issues, assisting in correct license usage and combination. Can create valid SPDX file(s) the original file had errors.
-	S15		Analyzes the compatibility of licenses at the element level (i.e., on specific clauses such as composition, attribution, or copyleft).
-	S31		Automated license term extraction system.
CaLi	S42		Orders licenses in terms of compatibility and compliance. To identify the compatibility among licenses, restrictiveness relation is enhanced with constraints.
EULAide	S43		Extracts information from license text to list permissions, prohibitions, and duties

* Studies listed in parentheses extend the original work.

5.2.4 Other studies

Studies categorized as other studies were either vague (S34, S40, S47, S51) or lacked validation (S6, S20, S37, S45, S52). These studies will be summarized here briefly.

Distributed release audit tool (DRAT) is an extension to the release audit tool (RAT) used by Apache project (Mattmann et al. 2015) (S40). As far as we are aware, RAT has not been described academically. RAT's primary function is to automatically audit the code and perform an open source license analysis focusing on source code headers. RAT can also be used to add license headers to source files. DRAT improves RAT's performance by distributing the workload. Kechagia et al. (2010) (S34) apply a "special Makefile target" to collect dependencies between FreeBSD applications. They visualize the results using the *GraphViz gvpr* tool. Also, they manually locate a key phrase that "uniquely identified" that license. With those signature phrases, they could search licenses on source files using *fgrep*. Unfortunately, these methods are not described in detail.

Singi et al. (2019) (S47) introduce CAG - Compliance Adherence and Governance framework that "uses blockchain technologies. The framework (i) enables the capturing of required data points based on compliance specifications, (ii) analyzes the events for non-conformant behavior through smart contracts, (iii) provides real-time alerts, and (iv) records and maintains an immutable audit trail of various activities." License compliance tools could be implemented as part of this framework.

Xu et al. (2010) (S51) propose open source license checker (OSLC) as a partial license-tracking tool that checks for license conflicts. Yun et al. (2017b) (S52) propose a license identification method based on the Levenshtein Distance algorithm and compatibility analysis of different licenses based on a license's "feature-points."

The Carneades software system provides support for arguments, that is, formal representations of facts, concepts, defeasible rules, and argumentation schemes. It allows constructing, evaluating, and visualizing of these arguments (Bavota et al. 2014; Gordon 2011) (S20 and S6). The OSS license compatibility analysis is then performed using structured argumentation (Gordon 2011).

Lokhman et al. (2012) (S37) present a tool, OSSLI, that addresses the legality concerns of open source at the level of software architecture. The tool focuses especially on validating architectural models against open source legality constraints and proposes remedial architectural solutions.

Pellegrini et al. (2019) (S45) describe the DALICC, a software framework that supports the resolution of licensing conflicts that occur in the reutilization of digital assets. However, it can be used for analyzing OSS licenses. DALICC provides a library of machine-readable standard licenses. In addition, it provides information about the equivalence, similarity, and compatibility of licenses.

5.2.5 Software Package Data Exchange

The reporting of the license analysis results has evolved significantly in the second decade of the millennium in SPDX format, even though none of the publications included in the review cycle address the format specifically. However, it has been mentioned or used in several studies (German and Di Penta 2012; Jaeger et al. 2017; Kapitsaki and Kramer 2015; Kapitsaki et al. 2017). SPDX is a "standard format for communicating the components, licenses, and copyrights associated with a software package" (SPDX Workgroup 2020a). Its goal is to document the license of a system and its files and their licenses in a well-defined format (SPDX Workgroup 2020a). SPDX was sent to the International Organization for Standardization (ISO) for consideration as a publicly available specification in August 2020 (SPDX Workgroup 2020b).

SPDX files appear in various formats. These include RDF (Resource Description Framework) files, a textual key-value pair format referred to as tag format, spreadsheet format, JSON format, and YAML format (SPDX Workgroup 2020a). These formats can be created manually, but preferably, these should be created automatically based on the results of automated license analysis tools. For instance, FOSSology is capable of creating SPDX files in multiple formats (Jaeger et al. 2017). The SPDX consortium also provides some tools that assist in the manipulation of SPDX files, including file converters and comparators (Kapitsaki and Kramer 2015).

The list of approved OSS licenses maintained by OSI is only a subset of the licenses currently in use in open source projects. The SPDX working group collects OSS licenses, including also those that are not approved by OSI. At the time of writing the current dissertation, this effort has identified more than 400 licenses. The SPDX workgroup also maintains list of commonly found exceptions to open source licenses that permits certain uses that would otherwise be forbidden (SPDX Workgroup 2018).

Some notable fields in the SPDX specification include, for example, *Declared License, Concluded License, All Licenses Information from Files*, and *Copyright Text*. *Declared License* lists the license(s) that have been announced by the authors of the package, whereas *Concluded License* indicates the license that the creator of the SPDX file concluded (Kapitsaki et al. 2017). These are typically identical, but the creator (person or an automated tool) of the SPDX file may have drawn different conclusions than the authors, so these may differ. *All Licenses Information from Files* is the list of all licenses that were found it the package. *Copyright Text* field's purpose is to identify the copyright holder of the file, as well as any dates present. This information assists in satisfying the documentation obligations of the license when a package is reused.

Licenses supported by SPDX can be used as single license identifiers (i.e., usage of one license). These single license identifiers are complemented by the SPDX License Expressions that provide more accurate information for the licensing terms of a software product (Kapitsaki et al. 2017). Expressions (e.g., AND, OR, WITH, and +) allow the indication of license exceptions and the combination

of license identifiers. For instance, the AND operator is used, when the software needs to comply with more than one license at the same time (e.g., GPLv3 AND MIT), whereas license exceptions are indicated using the WITH operator (e.g., GPLv3 WITH classpath-exception) (Kapitsaki et al. 2017).

5.3 Identification step of license compliance

As OSS reuse has become more and more popular, methods and automated tool support in the identification step of OSS license compliance has expanded beyond license and dependency analyses. Research has especially expanded to identify the origin of the reused snippet or package. Also, improvements in license identification and dependency analyses have been reported.

5.3.1 Identification of the origin of the OSS

From the included literature, we identified six papers (S8, S9, S24, S25, S46, S53) that describe validated features for identifying the origin of the OSS.

Ragkhitwetsagul and Krinke (2019) (S46) implemented Siamese, a clone search tool for Java source code. Siamese incorporates multiple code representation techniques to transform Java code into four code representations to detect different types of clones at once and a query reduction technique to automatically reduce the query size on the fly based on token document frequencies. Siamese is capable of detecting different types of clones: copy-paste clones (Type-1), clones with variable renaming (Type-2), clones with added or deleted statements (Type-3), and semantic clones or two code fragments with a different syntax but that share the same semantic (Type-4). Siamese requires the source code base (e.g., GitHub) to be processed before clone searching is possible as it needs to process a given source code base(s) to generate a searchable code index.

Inoue et al. (2012) (S24) propose an integrated approach to code history tracking for open source repositories and its prototype implementation: Ichi Tracker (Integrated Code History Tracker). Ichi Tracker uses source code search engines such as SPARS/R, Google Code Search, and Koders to find clones. It takes a code fragment as its query input and returns a set of cloned code fragments which are found by the search engines. Ichi Tracker extracts words from snippets, generates queries based on the keywords, analyzes the search results, filters false positive using the code clone detection tool CCFinder, and outputs the result. Ichi Tracker helps users to understand the backward and forward history of the queried code fragment.

OSS in Java applications is often reused using Maven package manager, and these packages do not necessarily include the actual source code but only compiled Java archives (JAR). To identify the source files of Java binary libraries, Software Bertillonage, an anchored signature matching technique, has been proposed (Davies et al. 2011, 2013) (S8, S9). Even though the papers describing the method

does not name the tool, the authors refer to this tool as Joa (German and Di Penta 2012). Davies et al. (2011, 2013) consider a list of attributes present in both the source and binary forms of Java classes such as class's name and namespace, inheritance tree, implemented interfaces, methods, modifiers, return types and method parameters, and relative position of methods in the class. From these attributes, a signature can be composed. To compare two archives, they define a metric called the similarity index of archives, which is intended to measure how similar two archives are with respect to the signatures of the classes within them. The inclusion index is the proportion of class signatures found in two compared archives. Given a binary archive (JAR file), they use the similarity and inclusion indexes to rank the likelihood that any archive in a corpus (Maven2 repository) might contain the same code found in the binary archive. Davies et al. find that they could identify the correct product information of contained binary Java archives if the product was present in Maven. However, if a product is present, they find that identifying the correct version could be tricky. Although Joa can extract the candidates of reused components, a user often has to manually identify the original components among the candidates. Ishio et al. (2016) (S25) propose an improvement to Joa: a method to automatically select the most likely origin of the components reused in a product. It is comprised of two steps: signature extraction and comparison based on these signatures. The method results in improved accuracy on the identification compared to the original method.

Zhang et al. (2010) (S53) describe LChecker, a tool they develop for the automatic checking of license compliance. LChecker utilizes the Google Code Search service to check whether the analyzed file exists in an OSS project. LChecker tokenizes the program and uses the resulting token list to perform queries via the Google Code Search service. Tokens are created by removing all comments, blank files, and headers in the program. LChecker then performs a syntactical analysis of the program and identifies unique lexical tokens. Tokens that are common to all programs are removed by LChecker before querying. The remaining tokens are concatenated into a single string, which is used as an input for Google Code Search API. If files already exist in the Internet as OSS programs, Google will return the OSS programs together with their license information. Validation of this tool include only one fairly small program (67 files), so the validity of the approach remains open. The paper also describes license identification and compatibility checking. However, as these are described very vaguely, they will not be presented further in the current study.

In addition to papers describing the new automated features presented above, there are seven (S7, S19, S33, S35, S39, S41, S50) other papers that describe other methods for the origin identification. Bei and Yuan (2013) (S7) propose a process of programmers' claims of the certificate of origin of source code and a workflow. It is based on series of questions that are raised when source code is committed to the source code management system. The usage of OSS in this stage raises a flag that the given source code needs further investigation to be accepted as part of the implemented system.

There are several studies that use clone detection tools in conjunction with

other tools to identify the origins and license of OSS. Golubev et al. (2020) (S19) conduct a study of possible block-level code borrowing and license violations in the Java corpus of GitHub. They use a method that combines two tools: SourcererCC to detect clones on the block level and a modified version of Ninka to detect licenses in files. Mlouki et al. (2016) (S41) use a combination of tools to identify the licenses of mobile applications. They use Ninka to identify licenses from source files, Joa to detect licenses form of JAR archives, and CCFinderX to detect code clones. Mathur et al. (2012) (S39) use a combination of Google Code repository and their listed licenses and clone detection method based on the plagiarism detection tool MOSS to identify potential licensing violations of OSS. Kashima et al. (2011) (S33) detect the license of code snippets used in copy-paste reuse. Their detection process employs Ninka for license detection and CCFinderX for copy-paste clone detection. Lee et al. (2015) (S35) use "machine-based algorithms to narrow down the potential violations and guide non-experts to manually inspect these violations." They develop an unnamed tool that could automatically detect cloned parts and use FOSSology to identify the licenses of these cloned parts. Their tool then identifies possible license inconsistencies between these cloned parts. That information can be used to automatically narrow down potential violations to reduce the workload. Wu et al. (2015) (S50) focus on detecting the license inconsistencies among file clones. They use CCFinder to analyze and determine whether two files are semantically identical and adopt Ninka to detect the license of the source file. Wu et al. group the semantically identical files and report the groups that contain license inconsistencies.

5.3.2 License identification

From the selected literature, we have identified nine papers (S10, S11, S17, S22, S23, S26, S36, S46, S49) that describe validated features or methods for identifying the license from the source files. Kapitsaki et al. (2015) (S28) also list and compare identification tools.

German et al. (2010b) (S17) describe the challenges of identifying the license under which the source code is made available, proposing a sentence-based matching algorithm to automatically do it and its implementation with a tool named Ninka.

They use license statements for the identification of the license and define these as license information, which is typically found in a comment at the very beginning of the file. A license statement typically contains four sections: 1) a list of copyright owners, 2) a list of authors (if different from the copyright owners), 3) the license or licenses that cover the file, and 4) warranty and liability statements (German et al. 2010b). The licenses in the licensing statement can be of two types (German et al. 2010b):

by-inclusion: the text of the license is embedded in the file

by-reference: the license statement indicates where the text of the license can be found

German et al. discover that these licensing statements are prone to errors. Errors

include, for example, files without license, copy pasting a wrong license statement (e.g., GPLv3+ was mistakenly used instead of GPLv2+), inconsistent license clauses, and incorrect name of the license.

German et al. also list challenges of license identification. These fall into three categories: "1) finding the license statement: how many licenses are in the file and where (within the file) they are located; 2) language related: the grammar, spelling, and wording use in a license statement; and 3) license customization: the terms of the licenses that are sometimes altered by their users" (German et al. 2010b).

Ninka uses a license identification algorithm that works by extracting the license statement from the file, breaks it apart into textual sentences, and proceeds to find a match for each one individually. The method requires a knowledge base of the following sets of information: 1) filtering keywords, 2) sets of equivalence phrases, 3) known sentence-token expressions, and 4) license rules.

One of the challenges of license identification using a sentence-matching technique is discriminating between those sentences that are part of the license statement and those that are not relevant. For this identification, Ninka uses a set of filtering keywords that need to be found from a sentence. If none of the keywords are found, sentence is not expected to contribute to the license of the file. Keywords can be composed of one or more words. Examples of filtering keywords are *license, conditions, disclaimer*, and *written permission*.

To deal with language-related challenges, Ninka includes sets of equivalent phrases. All phrases in a set are considered semantically equivalent. Equivalence phrase sets are, for example, as follows: *(at your option) any later version, any later version or any greater version; distributable, licensed, released,* or *made available*.

The third part of Ninka's knowledge base, the known sentence-tokens refer to a sentences of a known licenses. A license is a sequence of sentence-tokens, regardless of it is indicated by-inclusion or by-reference and they are generalized using one or more regular expressions. This set is used for translating each sentence found in the licensing statement into a sentence of a known license (a sentence-token).

Each license corresponds to a sequence of one or more sentence-tokens, which they call a license rule, which is the last part of Ninka's knowledge base. Identification of a license requires matching of one or more consecutive sentence-tokens. Typically, licenses indicated by-reference require only one consecutive sentence-token match, whereas by-inclusion licenses need more than one consecutive match.

Based on the knowledge base, Ninka uses an algorithm for license identification that is divided into six steps:

1. License statement extraction extracts the comments from the beginning of the file.
2. Text segmentation converts comments into a sequence of statements.
3. Equivalent phrase substitution scans each sentence and replaces it with a normalized version of the phrase if that is found.

4. Sentence filtering splits the sentence into two parts, one being the part that includes the legal keyword(s) and the other being the rest (if the legal part is empty, then the file does not have a license, so they label the source file NONE).
5. Sentence-token matching finds corresponding sentence-token for the legal part of the sentence (if no match exists, they use the sentence-token UNKNOWN)
6. License rule matching tries to match the sequence identified in the previous phase to license rules.

The license of the file is the list of licenses matched in the last step. If no license is matched, the file is considered to have an UNKNOWN license.

If Ninka does not have license identification rules for licenses, it reports them as "unknown license" that must be checked by developers manually. A license identification tool should be appropriately maintained by adding regular expressions corresponding to the new licenses, as new or derived OSS licenses appear nearly every year. To address this problem, Higashi et al. (2016, 2019) (S22, S23) construct a method to automatically create candidates of license rules to be added to a license identification tool such as Ninka.

FOSSology uses three license scan approaches to allow for more adaptive scanning and the ability to conclude licensing based on the results of these scanners (Jaeger et al. 2017) (S26). Nomos is the main license scanner in FOSSology and it uses regular expressions to identify licenses from the source files. Nomos holds more than 3000 snippets that map to more than 650 licenses (Jaeger et al. 2017). Jaeger et al. notes that disadvantage of matching license-relevant text findings with regular expressions of Nomos scanner is the lack of an ability to detect manipulated license text which adds new conditions for know licenses. For example, if a new condition is added to MIT license, then Nomos would still identify that as a MIT license. FOSSology introduced a new license scanner, Monk, in version 3 of the software to address these issues. This scanner finds license texts faster than the original Nomos scanner. Monk also considers the reference license text collection from the FOSSology database. Monk compares these texts with the found text in the files of the uploaded software component. Monk tokenizes the license reference texts and computes the Jaccard[3] Text Similarity Index, adding a weighting to the computed index. To help the user to identify the license, both the matching and the difference between stored license text and found text is highlighted. However, Monk only recognizes those licenses which are part of the FOSSology license database. For this reason, the Nomos and Monk agents complement each other: Nomos detects unknown licensing statements or license texts, whereas Monk can give very precise detection results for all known licenses. Ninka has also been integrated into FOSSology to complement Nomos' findings. FOSSology is also capable of reusing information from previously analysed uploads which reduce the time needed for a component analysis. For ex-

[3] Paul Jaccard developed the Jaccard index of similarity (he called it coefficient de communauté) and published it in 1901.

ample, when scanning new versions of software, the analysis can be limited to compare differences with an older version. FOSSology has the ability to export license analysis results in SPDX format. FOSSology also has the ability to import SPDX files, which helps in reducing the effort when analysing new versions of a software component. Import also allows to for displaying SPDX files in a human-readable format.

In addition to Ninka and FOSSology, Kapitsaki et al. (2015) list other source code license analyzers: ASLA (Tuunanen et al. 2009), OSLC (Xu et al. 2010) (S51), LIDESC[4], Ohloh, and what-license. Ohloh and what-license were online services, that are no longer available.

In addition to the techniques described above, we have identified other matching techniques from the included literature that were described in less detail. Siamese (Ragkhitwetsagul and Krinke 2019) (S46) identifies the software license in a software project using a two-step approach. First, to detect a license at the project level, the software attempts to find a license file (LICENSE or LICENSE.txt) from the GitHub projects root level and matches the contents with the license patterns in its database. Second, to identify the license of each Java source code file, Siamese reads a license statement from the beginning of each file and performs file-level pattern matching. The finer-grained file-level license is preferred when there is a conflict between the file-level and project-level license. If the Siamese cannot identify the license, it reports that a manual validation is needed. Siamese found a number of clone pairs between Stack Overflow and GitHub projects where the code were exactly matched but had different software licenses.

Vendome et al. (2017b) (S49) study methods of identifying license exceptions by relying on machine learning. Their study includes 51,754 systems from which the presence of license exceptions could be identified. Vendome et al. discover 14 different exception types across 298 files. Identifying these exceptions are vital, since they directly impact the way in which OSS can be reused. They use Ninka to extract the comment portions of the source file and define the set of heuristics to identify license exceptions from these comments. Heuristics here can be defined based on the known license exceptions listed in the SPDX exceptions list. Vendome et al. discover that the categorization of license texts and usage of machine learning classifiers provides accurate identification of these exceptions. Support Vector Machine and Random Forest classifiers are found to be especially efficient.

Liu et al. (2019) (S36) propose a learning-based method for predicting licenses on the source file level. This paper illustrates the complexity of license analysis because the authors seem to lack a detailed understanding of OSS licenses. They propose a method that predicts the license of a source file, one based on the license of that file itself and the files it is dependent on. However, Liu et al. do not take into account that files with incompatible licenses cannot be combined and that the license for these files cannot be predicted. Also, their algorithm fails

[4] LIDESC: Librock License Awareness System http://www.mibsoftware.com/librock/lidesc/ accessed Jan 18, 2021

in cases where a single copyleft-licensed file is combined with several files of permissive licenses. This paper also includes several claims about open source licenses that do not correspond with established knowledge. For instance, Liu et al. claim that both GPLv2 and GPLv3+ are weak copyleft licenses and their combination should be licensed as GPLv2. This is clearly not the case, since these licenses are strong incompatible copyleft licenses.

Di Penta et al. (2010a) (S10) propose an automatic approach to determine the license of JAR archives. The method combines the use of a code-search engine (Google Code Search) with the automatic classification of licenses found from the text files within the JAR archive. Their approach is composed of four steps: 1) extract files from the JAR archive, 2) identify and classify licenses contained in textual files (e.g., .txt or .html) using FOSSology, resulting in the declared license(s), 3) retrieve licensing info for Java files from Google Code Search, resulting inferred license(s), and 4) combine the different sources of information by comparing the declared and inferred licenses and determining the probable license for the archive.

Di Penta et al. (2010b) (S11) also propose an approach that automatically tracks changes in the licensing terms of a system. This is useful when comparing two different versions of the same software as the analyzer is able to focus only on the changed parts. Change tracking is performed by extracting the licensing statements and calculating their length. Statements that are changed between two versions of the software are analyzed using FOSSology.

5.3.3 Dependency identification

From the selected literature, we have identified two papers (S48, S13) that describe validated features for identifying the dependencies of the used software.

Van Der Burg et al. (2014) (S48) propose an approach to construct and analyze the Concrete Build Dependency Graph (CBDG) of a software system by tracing system calls that occur at build-time. CBDGs can be used to identify which source files of the client system are being used, which external components are being used, and how the client source code and external components are being combined. To create CBDGs, they record which files are read and written by each step in the build process by tracing the operating system calls while the build is executing. For the collection of these calls, they use *strace*[5]. This method can precisely (88-100%) identify files that have an impact on the client deliverables with little impact on the actual build process.

Dyck et al. (2018) (S13) describe OMP that uses the *maven-dependency-plugin* to resolve all the transitive dependencies ("OSS, which directly used OSS depends on", (Dyck et al. 2018, p. 44)) given by the POM of the Maven project. Information about a dependency is stored as a Maven artifact inside a dependency node, which represents the dependency as a graph. After having created a graph of all transitive dependencies, a pattern artifact filter is applied for all

[5] strace is a diagnostic, debugging, and instructional userspace utility for Linux. https://strace.io/ accessed Jan 18, 2021

dependencies for which no license information is needed, for example, internal libraries.

5.4 Approval of OSS reuse

The approval step of the license compliance process is composed of understanding the licensing of given OSS and, based on this understanding, approving or disapproving the reuse. Especially here, questions related to the compatibility of different licenses are relevant. Also, license comprehension is vital to understand what the licensing terms mean on a practical level.

5.4.1 License compatibility checking

From the selected literature, we have identified eight papers (S13, S14, S15, S16, S27, S30, S44, S48) that describe validated license compatibility checking features or methods.

Dyck et al. (2018) (S13) propose a dependency identification method described in Subsection 5.3.3. For each dependency identified, three artifacts are downloaded from the Maven repositories. These are the JAR file, the license text artifact containing the text of the license, and the license information artifact containing additional information about the license. OMP checks the rules that apply to the license types. If the rules are not satisfied, a message describing the violation is printed out and the build process is aborted. Some types of licenses have licensing constraints such as not handing out the source code. To enforce the rules, the OMP retrieves all used license types based on the license information artifacts of all used OSS. This list is then checked against an internal set of rules for violations.

SPDX Violation Tools (SPDX-VT) examines the SPDX files, ensuring correct license usage and combination (Kapitsaki et al. 2017) (S30). The initial version of the tool was introduced in Kapitsaki and Kramer (2015) (S27), and the improved version that can also handle input of multiple SPDX files can be found in Paschalides and Kapitsaki (2016) (S44). SPDX-VT assists in verifying that the correct license information is applied to the software package, identifies incompatibilities within the package, and proposes licenses that can be applied when combining licenses from different software (Kapitsaki et al. 2017). The SPDX violation check process comprises the following steps: 1) verify whether the SPDX file contains correct declared license(s) (see Subsection 5.2.5), 2) give information on problematic license combinations in case violations are found, and 3) if no violations are found, propose a number of alternate licenses that can be used for the package based on the license information in the SPDX file. Using the findings in the SPDX files, a license graph and the compatibility algorithm are used to examine and detect potential violations. The algorithm solves the graph reachability problem and is implemented using modified version of Breadth First Search. The

tool is also able to create valid SPDX files in case the original file(s) has errors (Paschalides and Kapitsaki 2016).

Gangadharan et al. (2012) (S15) use Open Digital Rights Language (ODRL) to implement the clauses of OSS licenses in a machine interpretable way and introduce an algorithm that analyzes the compatibility between OSS licenses. Although ODRL is a expression language for specifying rights over digital assets, it can be used for expressing a software license in a machine interpretable way. ODRL is based on a model for rights expression which comprises the following core entities and their relationships: 1) assets: resource being licensed, 2) rights: rules concerning permissions, constraints, requirements, and conditions, and 3) parties: information regarding the provider, consumer, broker, and so forth. Their formalized license compliance (FLC) algorithm analyzes the compatibility of licenses at the element level (i.e., on specific clauses such as composition, attribution, or copyleft). Any two software components can be combined if the licenses of these components are found compatible by the FLC algorithm.

Van Der Burg et al. (2014) (S48) describe a license compliance assessment method performed using five steps: extract the licenses of each file in each of the studied systems using Ninka, identify the declared license of the client deliverables by examining product documentation, generate a dependency graph (see Subsection 5.3.3), annotate the graph file nodes with license information, traverse the graph to identify the sources that are used to create the client deliverables, and mark the client deliverables that contain sources that are released under incompatible licenses as inconsistencies.

Eghan et al. (2019) (S14) introduce OntTAM, a trustworthiness assessment model for software library adoption. OntTAM supports the automated analysis and assessment of the quality attributes related to the trustworthiness of libraries and APIs in open source systems, providing developers with additional insights into the potential impact of reused libraries and APIs on the quality and trustworthiness of their project. One implementation of such trustworthiness is the semantic analysis of license compatibility. The implementation identifies license violation in case any of the components of a parent project is dependant on components with non-compatible licenses. In their case study, the license data of libraries are collected using Maven.

German et al. (2010a) (S16) propose a method to understand and identify licensing compatibility issues. It consists of three steps, that have the goal of 1) extracting declared licenses and dependencies from the distribution packages (Fedora .spec -files), 2) extracting and classifying actual licenses from the source code files, and 3) using the information extracted in steps 1 and 2 to detect licensing incompatibility issues. For each binary package, they execute the *rpmquery* command and obtain (i) general information (e.g. brief description of the package, the package version, and author name), (ii) the declared licenses(s) of that package (as indicated in the metadata), (iii) the list of components and libraries provided by the package, and (iv) the list of resources (components and libraries) the applications in the package require. Second, they extracted the files from the source package and then used Ninka to classify their licenses. The extracted

information was stored in a relational database. German et al. then queried this database to identify incompatibilities according to the compatibility rules defined in earlier studies.

5.4.2 License comprehension

From the selected literature, we have identified 13 papers (S1, S2, S3, S4, S5, S15, S26, S29, S31, S32, S38, S42, S43) that describe license comprehension-related features or methods.

Modeling all possible licenses and their relations is needed in order to decide which licenses are appropriate for a specific software system (Kapitsaki et al. 2015). Even though selecting an OSS license is outside the scope of the current work, the selection process requires similar license comprehension than compliance. *findOSSLicense* is a tool for selecting a license and is based on the modeling of OSS licenses (Kapitsaki and Charalambous 2016, 2019) (S29, S32). Kapitsaki and Charalambous propose a license modeling method that "captures different terms of the software license text, divided into rights, obligations, and additional properties covering the range of license terms of different licenses." They analyze 33 open source licenses and discover 38 generalized terms such as *MayCopy, MaySublicense, MustOfferSourceCode,* and *LimitedLiability* (Kapitsaki and Charalambous 2019). For the term extraction, Kapitsaki and Paschalides propose an automated extraction system in another paper (Kapitsaki and Paschalides 2017) (S31). The algorithm for term extraction is composed of 1) data preprosessing, including noise removal and sentence segmentation, 2) license term creation and mapping of these terms to a set of different phrases in OSS license texts, hence resulting a term to sentence map, 3) topic modeling: a statistical model for finding abstract topics in a collection of documents, and 4) term-to-topic matching. For topic modeling, they use the Latent Dirichlet Allocation (Blei et al. 2003) algorithm. For term-to-topic matching, they use the results from topic modeling the term to the sentence map. Actual matching uses a Cosine Similarity (Karypis et al. 2000) algorithm.

FOSSology provides an automated feature that assists in license comprehension. An administrator can define of obligations and risks, which can be associated with the licenses. When a license report is generated by FOSSOlogy, it will list all the obligations and risks of the licenses in effect (the concluded licenses) (Jaeger et al. 2017) (S26).

Moreau et al. (2019) (S42) propose CaLi, a model that partially orders licenses in terms of compatibility and compliance. In a license, actions can be distributed into status, for example, permissions, obligations, and prohibitions. To decide if a license is less restrictive than another one, it is necessary to know if an action in a status is considered less restrictive than the same action in another status. To identify the compatibility among licenses, Moreau et al. refine the restrictiveness relation with constraints.

Nejad et al. (2016) (S43) introduce the EULAide system, which comprises an information extraction pipeline tailored for license processing. It is composed of

the Ontology-Based Information Extraction (OBIE) method for license term and phrase extraction to facilitate a better understanding by humans. The pipeline that processes the license consists of 1) a linguistic pre-processing stage, 2) an ontology-based Gazetteer, and 3) the OBIE transducer. The first two steps create annotation sets that pair bits of text to the Open Digital Rights Language[6] elements. OBIE Transducer uses the previous annotation sets as inputs and matches pre-defined annotation patterns to the final annotation sets of permission, prohibition, and duty.

Alspaugh et al. (2012) (S3) present the results from an analysis directed toward a formal representation capable of covering an entire license. To form this type of representation, they identify the license's actions and relate them to the actions for exclusive rights defined in law and to the actions defined in other licenses. Actions are the most common constructs in their analyzed license, LGPLv2.1, and are essential in how the license is applied. The focus on actions that can appear in both a right and an obligation also leads to a more flexible and generalized approach for parameterizing actions and deriving a subsumption relation among them. They also discover, that everything in the LGPLv2.1 text may be classified as either 1) the definition of a term, 2) a right, 3) an obligation, 4) a modifier to a definition, right, or obligation, or 5) text without legal effect.

Alspaugh et al. (2010) (S1) define the challenges of heterogeneously licensed systems. These include, for example, what license applies to the resulting system where more than one license is used? What rights or obligations apply? How can one determine which license constraints match, include, or conflict with one another? To address this problem, Alspaugh et al. present a formal meta-model that presents the actors, rights, and obligations. They analyzed 10 licenses to empirically validate and improve their model. Since this model only lists conflicts, another paper (Alspaugh et al. 2011) (S2) enhances the model by presenting intellectual property results in terms of the arguments supporting them. As a result of these two papers, a meta model has been created, as shown in Figure 14 (Alspaugh et al. 2011). Alspaugh et al. (2013b) (S5) use the meta model to describe guidance for achieving a legally valid architectural component strategy while obtaining desired license rights in exchange for acceptable obligations. Alspaugh et al. (2013a) (S4) present a license analysis scheme to identify the license conflicts arising from composed software elements. The scheme includes actions and objects. An action is a verb or verb phrase describing what may, must, or must not be done with the object (e.g., source code). A license then may be expressed as a set of rights, with each right associated (in that license) with zero or more obligations that must be fulfilled to enjoy that right. Alspaugh et al. discover that with a formal specification of a software system's architecture, they can associate software license attributes with the system's components, connectors, and subsystem architectures and calculate the copyright rights and obligations for the system. Conflicting licenses can be identified using these obligations.

Manabe et al. (2014) (S38) analyze the license of a source package and the license of the files it contains. For this purpose, they create license-inclusion

[6] https://www.w3.org/ns/odrl/2/ accessed Jan 18, 2021

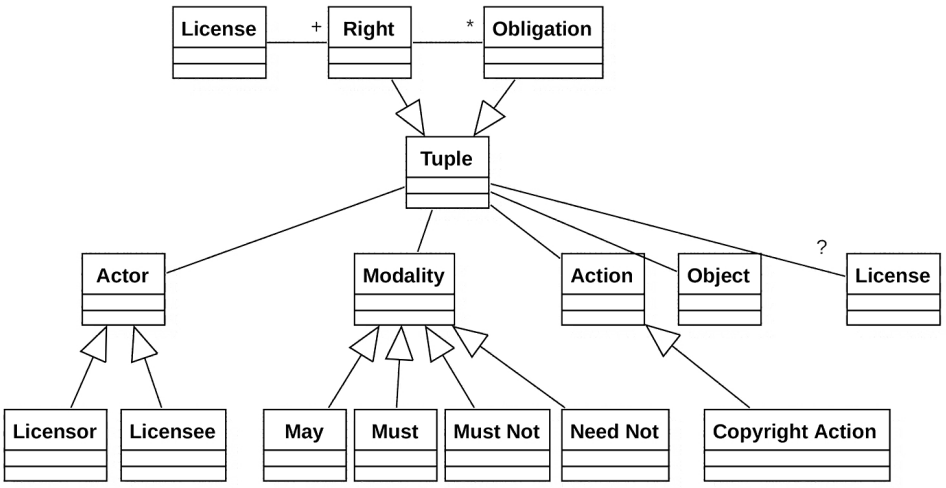

FIGURE 14 Meta model for licenses (Alspaugh et al. 2011).

graphs. First, they define the inclusion of one license into another license, such as BSD licensed code in a GPLv3 licensed package. Using the inclusion relationship, they could compute the licensing inclusion graph of a given collection of software packages. Second, they define the license inclusion graph of a package license as "the subgraph that ends in a given package license. All the edges in the graph which share a single destination node and come from different nodes with the same license are merged into a single edge." As an example, they present a package which is licensed under GPLv3 and includes 10 source files with BSD2 license. A graph representing this, includes two nodes "BSD" and "GPLv3" and a single edge from "BSD" to "GPLv3" with a weight of 10. Finally, they visualize the license-inclusion graphs to display the license of the package, the included licenses, and the proportions of each included license.

5.5 Satisfying the license obligations

From the selected literature, we have identified five papers (S12, S13, S17, S26, S51) that describe the features or methods related to satisfying the license obligations.

License identifiers typically list the used licenses, which is one of the key features when preparing the required documentation for the satisfaction step. Such tools include Ninka (S17), FOSSology (S26), and OSLC (S51). Also, the retrieval of full license text is often needed as it needs to be provided as part of the redistribution of the OSS. The literature regarding the retrieval of full license text is vague in many places. However, one example is found in OMP (S12, S13), which stores artifacts such as third-party dependencies and OSS license texts in the Nexus repository. When releasing software, relevant artifacts such as OSS source code and license texts are pulled from the Nexus repository and provided

together with the products (Dyck et al. 2018).

Research related to another key feature, acknowledgement documentation, is in its very early stages (Hemel 2015). Fully automated copyright extraction is very difficult as there is no standard notation for copyright statements, that is, copyright symbol, or the word "copyright" is not required (Hemel 2015). Also, copyright statements do not need to be in English. FOSSology attempts to fulfill this need as it strives to extract the relevant information. Also, FOSSology contains the ability to edit the copyright phrases found by the analysis process as it is sometimes necessary to post-process the results, mainly to remove formatting characters (Jaeger et al. 2017).

5.6 Validity

Based on the guidelines for performing a systematic literature review and mapping studies (Kitchenham et al. 2015; PRISMA 2015), we have systematically identified and addressed potential threats to the validity of our study by taking steps to minimize or mitigate them.

To ensure the internal validity of the systematic approach, the source selection is described in detail. To make sure that this review is repeatable, the search engines, search terms, and inclusion/exclusion criteria are carefully defined and reported. Potentially problematic issues are related to a limitation of search engines and search terms as limited engines and terms can lead to an incomplete set of studies. To mitigate the risk of finding all relevant studies, formal searching using defined keywords was conducted. Also, we performed a snowball search to ensure study coverage. Our automated search coverage was fairly successful as it resulted in 45 out of the 53 included studies. Applying inclusion/exclusion criteria can be biased based on the judgment and experience of the author. The selection validation was documented, showing substantial agreement between the author and supervisors of this thesis.

To ensure the validity of the data extraction and synthesis (i.e., construction of the results), it should be made sure that there is a clear link from the research questions to the data and then to the syntheses that answers those research questions (Kitchenham et al. 2015). The research questions related to the second cycle of the research process were designed to cover our goal, and these questions are answered accordingly. A threat to construct validity comes from the data extraction, classification, and empirical evidence of the studies (Kitchenham et al. 2015; PRISMA 2015). The methods of data extraction and classification were described in detail and followed accordingly. We extracted and classified the data from the studies according to these methods, ensuring that any bias of individual studies would be minimized. To identify papers with the most convincing empirical evidence, we extracted the validation of the proposed solution and references (i.e., which studies use proposed solution in their study) of that solution. Studies with limited validation or with no references in other papers were introduced only

briefly.

5.7 Summary

In this chapter, we have described the SLR related to automated OSS license compliance. The review consisted of 53 included publications. We identified user needs that were not present in the design cycle (see Table 9) and collected the automated features (see Table 10) that were described in the included publications.

6 DISCUSSION

In the previous chapters, we conducted a study related to automated OSS license compliance and did so in two cycles. In this chapter, we will summarize the results in accordance with our research questions and list our main contributions.

6.1 Revisiting research questions

We stated two main research questions related to automated license compliance:

RQ1 What are the user needs to fulfill automated open source license compliance?

RQ2 What software features are needed to fulfill the user needs?

These questions were answered separately for both of the two cycles of the current study. For the design cycle, we identified and listed the user needs that were published in Tuunanen et al. (2009). These include needs for every step of the license compliance process. A tool that supports these needs, ASLA, was implemented and reported (Tuunanen et al. 2006a,b, 2009). ASLA's functionality was validated using 12 different open source packages of different sizes (Tuunanen et al. 2009).

In the review cycle, we collected user needs that were not identified in the design cycle from the 53 studies that were included in our SLR. These needs are summarized in Table 9.

For the identification step of the compliance process, we recognized that the identification of the origin of the software was a new field that was not addressed in the design cycle. The identification step also includes license and dependency identification, techniques that was already present in the design cycle. The methods used for the identification of the origin of the software use clone detection techniques (e.g., Ragkhitwetsagul and Krinke (2019)). These techniques are not limited to identifying the origin of OSS as they are used, for example, for code quality assessment and the detection of code from binaries (Ain et al. 2019). Both

TABLE 11 Major challenges of license identification.

Type	Challenge
Finding the license statement.	License statements are usually mixed with other text.
	The comment characters and the various kinds of white space characters prevent exact matching.
	Files might reference another file where the license is located.
	Files might contain multiple licenses.
	Individual source files have no mention of used license.
Language related.	Licensing statements contain spelling errors.
	A given license is referred in different ways.
	Licensors change the spelling/grammar of the license statement.
License customization.	Several licenses must be customized when used.
	Licensors modify, add or remove conditions to well-known licenses.
	Licensors modify licenses for various intents.

cycles describe several challenges of license identification that highlight the need of sophisticated methods. These challenges of license identification are summarized in Table 11, here combining the identified challenges in both cycles (German et al. 2010b; Tuunanen et al. 2009). The review cycle includes similar regexp-based license identification techniques that were employed in ASLA in the design cycle and the sentence-matching method described by German et al. (2010b). Only two new techniques for dependency identification were introduced in the review cycle (Van Der Burg et al. (2014) and Dyck et al. (2018)). The review cycle included no papers describing techniques that would provide a possibility of manually determining the license of a source file or exclude particular files from license analysis. Both of these features were available in ASLA. Only one new automated method to add new license identification templates, a feature already present in ASLA, was described in the review cycle (Higashi et al. 2016, 2019). Also, the identification of the used parts of the OSS package, which was listed as one of the needs in the design cycle, is still relevant. Modern mix-and-match reuse relies heavily on the reuse of components of any shape and size (Mikkonen and Taivalsaari 2019). The installation of one reused package can fetch numerous additional packages (e.g., in case of the Maven or npm) that may not be used at all, so it is relevant to focus the license analysis only on the used parts. No studies in the review cycle address this issue.

Another new user need revealed by the review cycle is license comprehension, which is needed in the approval step of license compliance process. Based on the results of Almeida et al. (2019) and study conducted by Harutyunyan et al. (2019), the compliance tool should help users interpret open source licenses. The review cycle revealed several methods related to license comprehension: formal modeling of the licenses (e.g., Alspaugh et al. (2011)), implementations that extract information from license text (e.g., Kapitsaki and Paschalides (2017)), and automated tools that identify permissions, obligations, and prohibitions (e.g., Moreau et al. (2018)). Another field in the approval step, license compatibility checking, revealed several studies during the review cycle. These include approaches such as dependency-based compatibility checking similar to ASLA (e.g., Dyck et al. (2018)), a compatibility analysis based on SPDX files (e.g., Kapitsaki et al. (2017)), and license clause-level compatibility checking (Gangadharan et al. 2012).

Permissive licenses have gained popularity over the years, and it would have been expected that the research related to automated license compliance would follow this trend. A study from the early millennium by Capiluppi et al. (2003) shows that GPL was by far the most popular license, covering 77% of the 406 analyzed projects, when the most popular permissive license BSD, covered only 5% of the projects. Vendome et al. (2017a) study Java projects in GitHub and relative license usage of the analyzed projects between 2002 and 2012. The study shows that in 2002, permissive and copyleft licenses had about equal usage. However, in 2012, copyleft licenses had less than a 40% share of the total usage. In 2020, according to Goldstein (2020b), 67% of open source components had permissive licenses, MIT (27%) and Apache 2.0 (23%) being in the first and second place of most popular open source licenses, respectively, covering half of all open source projects. The license compliance features relevant for OSS licensed under permissive licenses are often related to documentation features, as permissive licenses main obligations are related to acknowledgement of the authors, whereas features such as a compatibility analysis are more relevant in case of being compliant with copyleft licenses.

Acknowledgement documentation along with offering of the source code in case of copyleft licenses are part of the satisfaction of the license terms, the third step of the license compliance process. Both cycles, however, produced the least amount of research related to this step. This is slightly surprising as increased OSS reuse along with the increased popularity of permissive licenses emphasizes the need of documentation features. To fulfill documentation obligations, a need for the formation of acknowledgement documentation that includes the extraction of copyright statements was identified in the review cycle. One example of an implementation that produces relevant documentation, that is, source code and full license text, was introduced in Dyck et al. (2018). However, automated methods for copyright extraction seems to be in its early stages. Hemel (2015) notes that a copyright extraction system that works better than current ones is needed. However, it is not expected to be fully automated but to use more context-sensitive information, such as author information from version control systems or other external sources of information (Hemel 2015).

As we compare the features presented in the two cycles, the strengths of ASLA that were not surpassed during the review cycle include a compatibility analysis based on the dependency map, the ability to maintain compatibility rules within a graphical user interface (Kapitsaki and Kramer 2015), and the detection of modified license text (Kapitsaki et al. 2015). Manual identification techniques and the ability to exclude program parts from the analysis are also yet to be implemented in other tools.

Only one tool that was present during the design cycle has evolved during the review cycle. This tool, FOSSology, is also the only programming language independent tool described in the review cycle that covers the whole license compliance process.

One notable field of software development was missing from the results of the review cycle. The included literature revealed no studies that would address

open source license compliance for web applications. For example, importing a single module into the Node.js application can fetch almost 1,000 dependencies (Morszczyzna 2017). To collect and understand the licensing restrictions and requirements of these modules would require automated support. However, the automated license compliance of these packages has not yet been described academically. This may be due to the fact that web applications are often not distributed but offered as a service; therefore the license identification and analysis is less relevant.

6.2 Contributions to theory and practice

The need for automated open source license compliance was identified in the design cycle. The need for an automated analysis is still relevant, as Harutyunyan et al. (2019) describe in their study of industry requirements for OSS governance tools: OSS license compliance is a central aspect and key tool requirement category within the companies they studied. Companies strive to automate license compliance, license scanning, and license management. Some companies employ continuous integration/deployment, thus requiring appropriate license compliance tools that can be integrated into their development process. Tool requirements for license compliance go on to encompass automated license interpretation, license identification, and documentation. In addition to these requirements, the tools need to identify the true origin of the software (e.g., determine who the real authors of software actually are) (Hemel 2015).

We listed detailed user needs for an automated license compliance tool in the design cycle (Tuunanen et al. 2009) and collected new user needs in the review cycle. Based on these two cycles, we combined and merged the user needs for an automated license compliance tool that are presented in Table 12. Most of the needs were already identified in the design cycle, for example, automated identification of the license from the source files and license compatibility analysis. The needs identified in the review cycle include the identification of the origin of the OSS (N2) (e.g., Ragkhitwetsagul and Krinke (2019)), identification of exceptions to known OSS licenses (N7) (Vendome et al. 2017b), needs related to comprehension of OSS licenses (N10) (e.g., Kapitsaki et al. (2015)), proposal of the alternate licenses (N11) (Kapitsaki et al. 2017), SPDX reporting format of the analysis results (N12) (e.g., German and Di Penta (2012)), formation of acknowledgement documentation (N15) (Hemel 2015), and integration of the license compliance features into development tools (N16) (Dyck et al. 2018).

We implemented and reported ASLA, a license identification tool in the design cycle of the research process (Tuunanen et al. 2006a,b, 2009). ASLA implements features that fulfill the needs identified during the design cycle (see Table 3). The review cycle, consisting of a SLR, provides a comprehensive view of the features implemented during the review cycle related to the automated license compliance process and are summarized in Table 10. As far as we are aware of,

TABLE 12 User needs for automated open source license compliance tool.

Need		Step†
N1*	Identification of the program modules that are included in the program within a particular environment.	I
N2	Identification of the origin of source files.	I
N3*	Automated license identification of source files.	I
N4*	Manual determination of the source code license.	I
N5*	The addition of license identification templates.	I
N6*	Dependency identification of program modules.	I
N7	Identification of exceptions to known licenses.	I
N8*	The visualization and definition of license compatibility rules.	A
N9*	The identification of licensing problems, such as use of incompatible licenses.	A
N10	Listing of rights, obligations, and restrictions of licenses found in the analyzed package.	A,S
N11	Proposal of alternate licenses that can be used for the package.	A
N12	Reporting of the license analysis results in SPDX format.	A
N13*	The definition of license compatibility rules.	A
N14*	Browsing of the results of the license analysis.	A
N15	Formation of acknowledgement documentation for each identified component including list of used licenses with full license text(s), attribution notices, and extracted copyright information.	S
N16	Integration of automated license compliance features into development tools such as package managers or continuous integration tools.	I,A,S

* Identified in the design cycle of the study.

† Refers to the step of the license compliance process: identification (I), approval (A), and satisfy (S).

no other study has addressed the user needs and features of automated license compliance at this scale.

7 CONCLUSION

The present dissertation introduced automated support for open source license compliance over the span of two decades. It stated two main research questions:

RQ1 What are the user needs to fulfill automated open source license compliance?
RQ2 What software features are needed to fulfill the user needs?

The answers to these questions were described in two cycles.

The design cycle introduced the the initial set of user needs, which were composed of 12 individual needs (see Table 2), and our reverse engineering approach, called ASLA, that fulfilled these user needs. ASLA was developed and reported according to the DSR approach. The main contributions include, for example, the implementation of automated license identification from the source files and automated license compatibility analysis.

In the review cycle, we described what automated tools are available in 2020, how user needs have evolved, and what features have been improved and discovered since the introduction of ASLA. The review cycle is composed of SLR that included 53 studies. The review cycle revealed new fields in OSS license compliance, such as the identification of the origin of the software and automated license comprehension, that were not present in the design cycle. Also, we systematically collected new features related to license compliance from the included studies. A summary of these features is displayed in Table 10.

Based on the information of these two cycles, we merged and listed a set of user needs, which were composed of 16 individual needs (see Table 12) and which cover the whole open source license compliance process. It became evident that no tool is available that would support all of these needs. License identification and a compatibility analysis are fields that have the most mature solutions in the license compliance process. Also, clone detection, which helps identify the origin of the source code, and license comprehension are described in several studies, even though the features related to these are yet to be integrated into other compliance tools.

We discovered the following issues to be addressed in future research: 1) integrating the features that are scattered between different tools into one tool that can seamlessly integrate these different tools as part of development process and 2) improve automated methods to create required acknowledgement documentation that seems to be in its early stages (e.g., by extracting copyright information from source files). Also, features related to the new user needs identified in the review cycle such as clone detection and license comprehension have yet to be integrated into tools such as FOSSology (Jaeger et al. 2017) or OMP (Dyck et al. 2018) that cover the whole license compliance process. There are also only a few automated solutions that integrate license compliance features into development tools. Support for this is especially relevant in modern mix-and-match development, where snippets from development forums or packages downloaded by package managers are routinely integrated into applications as an integral part of the development process.

YHTEENVETO (SUMMARY IN FINNISH)

Tämä tutkielma pyrkii esittämään kattavan kuvan automatisoiduista toiminnallisuuksista ja metodeista, jotka tukevat avoimen lähdekoodin lisenssien noudattamista. Avoin lähdekoodi viittaa tietokoneohjelmistojen kehittämismalliin, jossa ohjelmistojen lähdekoodi asetetaan julkisesti saataville. Malli mahdollistaa myös ohjelmistojen vapaan käyttämisen sekä mahdollisuuden niiden muokkaamiseen, laajentamiseen ja uudelleenjakeluun. Nämä toiminnot ovat mahdollisia avoimen lähdekoodin lisenssien ansiosta. Lisenssit myöntävät oikeuksia, jotka olisivat muuten kansainvälisten immateriaalioikeuslakien perusteella kiellettyjä.

Lisenssit asettavat oikeuksien lisäksi uudelleenkäytölle myös eri asteisia ehtoja. Nämä ehdot voivat muodostua ongelmaksi esimerkiksi siksi, että joidenkin avoimen lähdekoodin lisenssien ehdot ovat keskenään ristiriitaisia. Tätä kutsutaan lisenssien epäyhteensopivuudeksi, jonka vuoksi kahta eri lisenssin alaista ohjelmistoa ei voida laillisesti yhdistää. Toinen, erityisesti kaupalliseen uudellenkäyttöön liittyvä ongelma on joidenkin lisenssien asettama ehto, jonka mukaan uudellenkäytettävään ohjelmistoon pohjautuvat ohjelmistot tulee julkaista saman avoimen lähdekoodin lisenssin alaisuudessa. Jotta ohjelmistoja voidaan tehokkaasti uudelleenkäyttää, tarvitaan automatisoitua lisenssianalyysiä. Analyysillä varmistetaan, että ohjelmistojen lisenssejä noudatetaan uudellenkäytön yhteydessä. Automatisointia tarvitaan esimerkiksi siksi, että ohjelmistot voivat sisältää satoja tai jopa tuhansia lähdekooditiedostoja, jotka on mahdollisesti lisensoitu eri tavoilla.

Tutkielman teoreettinen viitekehys (luvut 2-3) sisältää immateriaalioikeuksien esittelyn ohjelmistojen lisenssien näkökulmasta, avoimen lähdekoodin lisenssien esittelyn sekä avoimen lähdekoodin lisenssien noudattamiseen liittyvän prosessin kuvauksen. Prosessi sisältää kolme vaihetta: tunnista, hyväksy ja täytä lisenssiehdot. Empiirinen osuus (luvut 4-6) koostuu kahdesta syklistä: suunnittelusyklistä (Design cycle) ja katsaussyklistä (Review cycle). Suunnittelusyklissä toteutettiin ASLA (Automated Software License Analyzer) -niminen analyysityökalu. Tulokset julkaistiin kolmessa tieteellisessä artikkelissa vuosina 2006 – 2009. Katsaussykli muodostuu systemaattisesta kirjallisuuskatsauksesta, jossa kuvataan kuinka automatisoidut työkalut ja menetelmät ovat kehittyneet ASLA:n julkaisemisen jälkeen. Kirjallisuuskatsaukseen sisällytetyt artikkelit ovat vuosilta 2010 – 2020.

Suunnittelusyklissä tunnistettiin 12 käyttäjätarvetta, jotka liittyvät automatisoituun lisenssien noudattamiseen. Nämä sisältävät esimerkiksi lisenssien tunnistamisen lähdekooditiedostoista ja lisenssien yhteensopivuusanalyysin. Käyttäjätarpeiden pohjalta kehitettiin niitä tukeva ohjelmisto ASLA. ASLA validoitiin analysoimalla 12 avoimen lähdekoodin ohjelmistoa. ASLA:n avulla kyettiin automaattisesti tunnistamaan lisenssien yhteensopivuusongelmia, koska se tunnistaa lähdekooditiedostojen lisenssit ja niiden väliset riippuvuudet ja se sisältää lisenssien väliset yhteensopivuussäännöt. Lisenssien tunnistamisessa pyrittiin mahdollisimman suureen kattavuuteen. Tämä kattavuus voi kuitenkin vaihdella

merkittävästi, sillä useat ohjelmistot sisältävät ns. lisensointivikoja, kuten esimerkiksi puuttuvia lisenssitietoja. Tavoitteena oli tunnistaa jokaisen lähdekooditiedoston lisenssi käyttämällä ensin automatisoituja tekniikoita ja täydentäen analyysiä interaktiivisilla tekniikoilla. Tunnistamisen automatisoidussa vaiheessa pystyttiin tunnistamaan lähdekooditiedostojen lisenssi 1–97 %:ssa tapauksista. Tulokset kohentuivat 75–98 %:iin käyttäen interaktiivisia tekniikoita.

Katsaussykliin valikoitui 53 tieteellistä artikkelia. Näistä artikkeleista tunnistettiin yhteensä seitsemän uutta käyttäjätarvetta, joita ei ollut tunnistettu ensimmäisessä syklissä. Näitä ovat esimerkiksi avoimen lähdekoodin ohjelmiston alkuperän tunnistaminen ja lisenssiehtojen ymmärtämisen tukeminen. Erityisesti ohjelmiston alkuperän tunnistaminen on tärkeää, sillä uudelleenkäyttö on lisääntynyt merkittävästi viimeisen 10 vuoden aikana ja lähdekoodia kopioidaan usein ohjelmistosta toiseen. Alkuperän tunnistamisen avulla voidaan varmistaa ohjelmiston oikea alkuperä ja näin ollen myös sen lisenssi. Lisäksi lisenssit ovat lakiteknistä tekstiä, joten niiden ymmärtäminen on ohjelmistokehittäjille vaikeaa. Lisenssien ymmärtämisen tukemiseksi on esitelty menetelmiä, joilla lisenssien antamat oikeudet ja velvollisuudet voidaan esittää helpommin ymmärrettävässä muodossa. Käyttäjätarpeiden tunnistamisen lisäksi listataan automaattista lisenssianalyysiä tukevia toiminnallisuuksia, jotka on esitelty suunnittelusyklin jälkeen. Nämä on ryhmitelty ja listattu yllä mainitun lähdekoodin lisenssien noudattamiseen liittyvän prosessin mukaan.

Yhteenvetona voidaan todeta, että automatisoidulle lisenssianalyysille on selkeä tarve. Kahden tutkimyssyklin pohjalta yhdistettin ja listattiin yhteensä 16 itsenäistä käyttäjätarvetta, jotka kuvaavat kokonaisvaltaisesti automatisoituun lisenssien noudattamiseen liittyvät käyttäjätarpeet. Tutkimuksessa kävi selväksi, että mikään yksittäinen työkalu ei tue kaikkia näitä tarpeita. Lisenssien tunnistamiseen ja yhteensopivuusanalyysiin liittyvät toiminnallisuudet ovat parhaiten tuettuja. Jatkotutkimusta tarvitaan erityisesti parantamaan toiminnallisuuksia, jotka liittyvät tekijänoikeustietojen keräämiseen lähdekoodista ja olemassa olevien toiminnallisuuksien liittämiseksi osaksi työkaluja, joita käytetään päivittäisessä ohjelmistokehityksessä.

BIBLIOGRAPHY

Abadie, B. and Ledru, S. 2020. Engineering code quality in the Firefox browser: A look at our tools and challenges. ⟨URL: https://hacks.mozilla.org/2020/04/code-quality-tools-at-mozilla/⟩.

Ain, Q. U., Butt, W. H., Anwar, M. W., Azam, F., and Maqbool, B. 2019. A systematic review on code clone detection. IEEE Access 7, 86121–86144.

Aksulu, A. and Wade, M. R. 2010. A comprehensive review and synthesis of open source research. Journal of the Association for Information Systems 11 (11), 6.

Al-Zubidy, A. 2017. The search phase of software engineering systematic literature review: barriers and solutions. PhD thesis. University of Alabama Libraries.

Almeida, D. A., Murphy, G. C., Wilson, G., and Hoye, M. 2019. Investigating whether and how software developers understand open source software licensing. Empirical Software Engineering 24 (1), 211–239.

Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. 2011. Presenting Software License Conflicts through Argumentation. In Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering. Knowledge Systems Institute Graduate School, 509–514.

Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. 2013a. Software licenses, open source components, and open architectures. In Aligning Enterprise, System, and Software Architectures. IGI Global, 58–79.

Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. 2013b. The challenge of heterogeneously-licensed systems in open architecture software ecosystems. In Software Ecosystems. Edward Elgar Publishing.

Alspaugh, T. A., Scacchi, W., and Asuncion, H. U. 2010. Software licenses in context: The challenge of heterogeneously-licensed systems. Journal of the Association for Information Systems 11 (11), 2.

Alspaugh, T. A., Scacchi, W., and Kawai, R. 2012. Software licenses, coverage, and subsumption. In 2012 Fifth IEEE International Workshop on Requirements Engineering and Law (RELAW). IEEE, 17–24.

An, L., Mlouki, O., Khomh, F., and Antoniol, G. 2017. Stack overflow: a code laundering platform? In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 283–293.

Apache Software Foundation 2019. GPL compatibility. ⟨URL: https://www.apache.org/licenses/GPL-compatibility.html⟩.

Apache Software Foundation 2020. Apache License, Version 2.0. ⟨URL: https://www.apache.org/licenses/LICENSE-2.0⟩.

Asay, M. 2018. Who really contributes to open source. ⟨URL: https://www.infoworld.com/article/3253948/who-really-contributes-to-open-source.html#tk.twt_ifw⟩.

Azhakesan, A. and Paulisch, F. 2020. Sharing at scale: an open-source-software-based license compliance ecosystem. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, 130–131.

Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., and Lopes, C. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, 681–682.

Banaeianjahromi, N. and Smolander, K. 2016. What do we know about the role of enterprise architecture in enterprise integration? A systematic mapping study. Journal of Enterprise Information Management.

Bavota, G., Ciemniewska, A., Chulani, I., De Nigro, A., Di Penta, M., Galletti, D., Galoppini, R., Gordon, T. F., Kedziora, P., et al. 2014. The market for open source: An intelligent virtual open source marketplace. In 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 399–402.

Bei, L. and Yuan, S. 2013. Software Intellectual Property Management through Self-Claiming of the Certificate of Origin of the Source Code. In 2013 International Conference on Computational and Information Sciences. IEEE, 613–615.

Blei, D. M., Ng, A. Y., and Jordan, M. I. 2003. Latent dirichlet allocation. Journal of machine Learning research 3 (Jan), 993–1022.

Blue Oak Council 2020a. License List. ⟨URL: https://blueoakcouncil.org/list⟩.

Blue Oak Council 2020b. The Blue Oak Guide to Copyleft. ⟨URL: https://blueoakcouncil.org/copyleft⟩.

Boehm, B. 1989. Software risk management. In European Software Engineering Conference. Springer, 1–19.

Boughanmi, F. 2010. Multi-language and heterogeneously-licensed software analysis. In 2010 17th working conference on reverse engineering. IEEE, 293–296.

Capiluppi, A., Lago, P., and Morisio, M. 2003. Characteristics of open source software projects. In Proc. 7th European Conference on Software Maintenance and Reengineering (CSMR 2003), 317–330.

Clark, D. 2015. OSS Attribution Best Practices. ⟨URL: https://www.nexb.com/blog/oss_attribution_obligations.html⟩.

Creative Commons 2020. Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0). ⟨URL: https://creativecommons.org/licenses/by-sa/3.0/⟩.

Davies, J., German, D. M., Godfrey, M. W., and Hindle, A. 2011. Software bertillonage: finding the provenance of an entity. In Proceedings of the 8th working conference on mining software repositories, 183–192.

Davies, J., German, D. M., Godfrey, M. W., and Hindle, A. 2013. Software bertillonage: Determining the provenance of software development artifacts. Empirical Software Engineering 18 (6), 1195–1237.

Debian 2004. Debian Social Contract. ⟨URL: https://www.debian.org/social_contract⟩.

Determann, L. 2006. Dangerous Liaisons-Software Combinations as Derivative Works-Distribution, Installation, and Execution of Linked Programs under Copyright Law, Commercial Licenses, and the GPL. Berkeley Tech. LJ 21, 1421.

Di Penta, M., German, D. M., and Antoniol, G. 2010a. Identifying licensing of jar archives using a code-search approach. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, 151–160.

Di Penta, M., German, D. M., Guéhéneuc, Y.-G., and Antoniol, G. 2010b. An exploratory study of the evolution of software licensing. In 2010 ACM/IEEE 32nd International Conference on Software Engineering. Vol. 1. IEEE, 145–154.

Dorner, M., Capraro, M., and Barcomb, A. 2020. Quo Vadis, Open Source? The Limits of Open Source Growth. arXiv preprint arXiv:2008.07753.

Duan, R., Bijlani, A., Xu, M., Kim, T., and Lee, W. 2017. Identifying open-source license violation and 1-day security risk at large scale. In Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security, 2169–2185.

Dyck, S., Haferkorn, D., Kerth, C., and Schoebel, A. 2018. Automating Open Source Software License Information Generation in Software Projects. Journal of Systemics, Cybernetics and Informatics 16 (5), 44–49.

Dyck, S., Haferkorn, D., and Sander, J. 2016. An Organizational-Technical Concept to Deal with Open Source Software License Terms. In Proceedings of the 20th World Multi-Conference on Systemics, Cybernetics and Informatics, WMSCI, 5–8.

Ebert, C. 2008. Open source software in industry. IEEE Software 25 (3), 52–53.

Eghan, E. E., Alqahtani, S. S., Forbes, C., and Rilling, J. 2019. API trustworthiness: an ontological approach for software library adoption. Software Quality Journal 27 (3), 969–1014.

European Patent Office 2000. The European Patent Convention. ⟨URL: https://www.epo.org/law-practice/legal-texts/html/epc/2016/e/ar52.html⟩.

Evans, D. S. and Layne-Farrar, A. 2004. Software patents and open source: The battle over intellectual property rights. Va. JL & Tech. 9, 1.

Fedora 2013a. Licensing Guidelines. ⟨URL: https://docs.fedoraproject.org/en-US/packaging-guidelines/LicensingGuidelines/⟩.

Fedora 2013b. Licensing:Main. ⟨URL: https://fedoraproject.org/wiki/Licensing:Main⟩.

Fendt, O. and Jaeger, M. C. 2019. Open source for open source license compliance. In IFIP International Conference on Open Source Systems. Springer, 133–138.

Feng, M., Mao, W., Yuan, Z., Xiao, Y., Ban, G., Wang, W., Wang, S., Tang, Q., Xu, J., Su, H., et al. 2019. Open-source license violations of binary software at large scale. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 564–568.

Fershtman, C. and Gandal, N. 2007. Open source software: Motivation and restrictive licensing. International Economics and Economic Policy 4 (2), 209–225.

Finnish Institute for Health and Welfare 2020. An application that helps break the chains of infection Koronavilkku. ⟨URL: https://thl.fi/en/web/infectious-diseases-and-vaccinations/what-s-new/coronavirus-covid-19-latest-updates/transmission-and-protection-coronavirus/contact-tracing-app-will-help-stop-chains-of-infection⟩.

Fitzgerald, B. 2006. The transformation of open source software. MIS quarterly, 587–598.

Fosfuri, A., Giarratana, M. S., and Luzzi, A. 2008. The penguin has entered the building: The commercialization of open source software products. Organization science 19 (2), 292–305.

FOSSA 2020. Software Licenses in Plain English. ⟨URL: https://tldrlegal.com⟩.

Franch, X., Susi, A., Annosi, M. C., Ayala, C., Glott, R., Gross, D., Kenett, R., Mancinelli, F., Ramsany, P., Thomas, C., et al. 2013. Managing risk in open source software adoption. In ICSOFT 2013: Proceedings of the 8th International Joint Conference on Software Technologies, 258–264.

Free Software Foundation 2007. GNU Affero General Public License. ⟨URL: https://www.gnu.org/licenses/agpl-3.0.en.html⟩.

Free Software Foundation 2009. GCC runtime library exception. ⟨URL: https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html⟩.

Free Software Foundation 2016. GNU Hurd/ history. ⟨URL: https://www.gnu.org/software/hurd/history.html⟩.

Free Software Foundation 2019. What is free software? ⟨URL: https://www.gnu.org/philosophy/free-sw.en.html⟩.

Free Software Foundation 2020a. Frequently Asked Questions about the GNU Licenses. ⟨URL: https://www.gnu.org/licenses/gpl-faq.en.html⟩.

Free Software Foundation 2020b. Various Licenses and Comments about Them. ⟨URL: https://www.gnu.org/licenses/license-list.en.html⟩.

Free Software Foundation 2020c. What is Copyleft? ⟨URL: https://www.gnu.org/copyleft/⟩.

Free Software Foundation, I., Center, S. F. L., Gingerich, D., Sebro Jr, A. K., Kuhn, B. M., and Legal, C. 2013. Detailed Analysis of the GNU GPL and Related Licenses. ⟨URL: https://copyleft.org/guide/comprehensive-gpl-guidepa1.html⟩.

Freibrun, E. 1995. Intellectual Property Rights in Software – What They Are and How to Protect Them. ⟨URL: https://freibrunlaw.com/intellectual-property-rights-software-protect/⟩.

Fujita, K. and Tsukada, Y. 2012. An approach to the formal analysis of license interoperability. Computers & Electrical Engineering 38 (6), 1670–1686.

Galler, B. A. 1995. Software and intellectual property protection: copyright and patent issues for computer and legal professionals. Greenwood Publishing Group.

Gamalielsson, J. and Lundell, B. 2017. On licensing and other conditions for contributing to widely used open source projects: an exploratory analysis. In Proceedings of the 13th International Symposium on Open Collaboration, 1–14.

Gangadharan, G., D'Andrea, V., De Paoli, S., and Weiss, M. 2012. Managing license compliance in free and open source software development. Information Systems Frontiers 14 (2), 143–154.

Garcia, V. C., Lucrédio, D., Durão, F. A., Santos, E. C. R., Almeida, E. S. de, Mattos Fortes, R. P. de, and Lemos Meira, S. R. de 2006. From specification to experimentation: A software component search engine architecture. In International Symposium on Component-Based Software Engineering. Springer, 82–97.

Garousi, V. and Mäntylä, M. V. 2016. When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology 76, 92–117.

Georgievski, G. 2020. Intellectual Property Rights in Software. ⟨URL: https://www.odilaw.com/intellectual-property-rights-in-software/⟩.

German, D. M. and Di Penta, M. 2012. A method for open source license compliance of java applications. IEEE software 29 (3), 58–63.

German, D. M., Di Penta, M., and Davies, J. 2010a. Understanding and auditing the licensing of open source software distributions. In 2010 IEEE 18th International Conference on Program Comprehension. IEEE, 84–93.

German, D. M. and Hassan, A. E. 2009. License integration patterns: Addressing license mismatches in component-based development. In 2009 IEEE 31st International Conference on Software Engineering. IEEE, 188–198.

German, D. M., Manabe, Y., and Inoue, K. 2010b. A sentence-matching method for automatic license identification of source code files. In Proceedings of the IEEE/ACM international conference on Automated software engineering, 437–446.

Germonprez, M., Young, B., Mathiassen, L., Kendall, J. E., Kendall, K. E., Warner, B., and Cao, L. 2012. Risk mitigation in corporate participation with open source communities: protection and compliance in an open source supply chain. Risk 12, 15–2012.

GitHub 2020. The 2020 State of the Octoverse. ⟨URL: https://octoverse.github.com/⟩.

Goldstein, A. 2019. Open Source Licenses Explained. ⟨URL: https://resources.whitesourcesoftware.com/blog-whitesource/open-source-licenses-explained⟩.

Goldstein, A. 2020a. Everything You Wanted to Know About Open Source Attribution Reports. ⟨URL: https://resources.whitesourcesoftware.com/blog-whitesource/open-source-attribution-reports⟩.

Goldstein, A. 2020b. Open Source Licenses in 2020: Trends and Predictions. ⟨URL: https://resources.whitesourcesoftware.com/blog-whitesource/top-open-source-licenses-trends-and-predictions⟩.

Golubev, Y., Eliseeva, M., Povarov, N., and Bryksin, T. 2020. A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub. In Proceedings of the 17th International Conference on Mining Software Repositories. MSR '20. Association for Computing Machinery, 54–64. ISBN: 9781450375177. DOI: 10.1145/3379597.3387455.

Gordon, T. F. 2011. Analyzing open source license compatibility issues with Carneades. In Proceedings of the 13th International Conference on Artificial Intelligence and Law, 51–55.

Gordon, T. F. 2014. A Demonstration of the MARKOS License Analyser. Computational Models of Argument: Proceedings of COMMA 2014 266, 461.

Greene, T. C. 2001. Ballmer: 'Linux is a cancer'. ⟨URL: https://www.theregister.com/2001/06/02/ballmer_linux_is_a_cancer/⟩.

Haddad, I. 2018. Open Source Compliance in the Enterprise 2nd Edition. The Linux Foundation.

Haddad, I. 2019. Recommended Open Source Compliance Practices for the Enterprise. The Linux Foundation.

Hall, B. H. and MacGarvie, M. 2010. The private value of software patents. Research Policy 39 (7), 994–1009.

Hammouda, I., Mikkonen, T., Oksanen, V., and Jaaksi, A. 2010. Open source legality patterns: architectural design decisions motivated by legal concerns. In Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments, 207–214.

Hanson, M. 2020. Microsoft admits it was wrong about Linux and open source. ⟨URL: https://www.techradar.com/news/microsoft-admits-it-was-wrong-about-linux-and-open-source⟩.

Hartmann, B., Doorley, S., and Klemmer, S. R. 2008. Hacking, mashing, gluing: Understanding opportunistic design. IEEE Pervasive Computing 7 (3), 46–54.

Harutyunyan, N., Bauer, A., and Riehle, D. 2019. Industry requirements for FLOSS governance tools to facilitate the use of open source software in commercial products. Journal of Systems and Software 158, 110390.

Harutyunyan, N. and Riehle, D. 2019a. Getting started with open source governance and compliance in companies. In Proceedings of the 15th International Symposium on Open Collaboration, 1–10.

Harutyunyan, N. and Riehle, D. 2019b. Industry best practices for open source governance and component reuse. In Proceedings of the 24th European Conference on Pattern Languages of Programs, 1–14.

Hashimoto, G. and Portner, C. 2020. The Intersection of Trademarks and Open Source. ⟨URL: https://www.hopkinscarley.com/blog/client-alerts-blogs-updates/post/the-intersection-of-trademarks-and-open-source⟩.

Hazen, T. L. 1986. Contract Principles as a Guide for Protecting Intellectual Property Rights in Computer Software: The Limits of Copyright Protection, the Evolving Concept of Derivative Works, and the Proper Limits of Licensing Arrangements. UC Davis L. Rev. 20, 105.

Heirendt, L., Arreckx, S., Trefois, C., Yarosz, Y., Vyas, M., Satagopam, V. P., Schneider, R., Thiele, I., and Fleming, R. M. 2017. ARTENOLIS: Automated Reproducibility and Testing Environment for Licensed Software. arXiv preprint arXiv:1712.05236.

Hellstadius, Å. 2010. Software Patents. In Information & communication technology : legal issues. Jure, 362–396.

Hemel, A. 2015. Tooling for Open Source Software License Compliance. Computer Law Review International 16 (4), 101–106.

Hemel, A., Kalleberg, K. T., Vermaas, R., and Dolstra, E. 2011. Finding software license violations through binary code clone detection. In Proceedings of the 8th Working Conference on Mining Software Repositories, 63–72.

Hevner, A. R. 2007. A three cycle view of design science research. Scandinavian journal of information systems 19 (2), 4.

Higashi, Y., Manabe, Y., and Ohira, M. 2016. Clustering OSS License Statements Toward Automatic Generation of License Rules. In 2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP). IEEE, 30–35.

Higashi, Y., Ohira, M., Kashiwa, Y., and Manabe, Y. 2019. Hierarchical Clustering of OSS License Statements toward Automatic Generation of License Rules. Journal of Information Processing 27, 42–50. DOI: 10.2197/ipsjjip.27.42.

Iivari, J. 2007. A paradigmatic analysis of Information Systems as a design science, forthcoming in Scandinavian Journal of Information Systems. Draft 27p, ask the newest version from the author juhani. iivari@ oulu. fi.

Inoue, K., Sasaki, Y., Xia, P., and Manabe, Y. 2012. Where does this code come from and where does it go?—Integrated code history tracker for open source systems. In 2012 34th International Conference on Software Engineering (ICSE). IEEE, 331–341.

Ishio, T., Kula, R. G., Kanda, T., German, D. M., and Inoue, K. 2016. Software ingredients: Detection of third-party component reuse in java software release. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 339–350.

Ishio, T., Sakaguchi, Y., Ito, K., and Inoue, K. 2017. Source file set search for clone-and-own reuse analysis. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 257–268.

Jaeger, M. C., Fendt, O., Gobeille, R., Huber, M., Najjar, J., Stewart, K., Weber, S., and Wurl, A. 2017. The FOSSology Project: 10 Years Of License Scanning. IFOSS L. Rev. 9, 9.

Jones, T. C. 1984. Reusability in Programming: A Survey of the State of the Art. IEEE Transactions on Software Engineering (5), 488–494.

Kapitsaki, G. M. and Charalambous, G. 2016. Find your open source license now! In 2016 23rd Asia-Pacific Software Engineering Conference (APSEC). IEEE, 1–8.

Kapitsaki, G. M. and Charalambous, G. 2019. Modeling and recommending open source licenses with findOSSLicense. IEEE Transactions on Software Engineering.

Kapitsaki, G. M. and Kramer, F. 2015. Open source license violation check for spdx files. In International Conference on Software Reuse. Springer, 90–105.

Kapitsaki, G. M., Kramer, F., and Tselikas, N. D. 2017. Automating the license compatibility process in open source software with SPDX. Journal of Systems and Software 131, 386–401.

Kapitsaki, G. M. and Paschalides, D. 2017. Identifying terms in open source software license texts. In 2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 540–545.

Kapitsaki, G. M., Tselikas, N. D., and Foukarakis, I. E. 2015. An insight into license tools for open source software systems. Journal of Systems and Software 102, 72–87.

Karypis, M. S. G., Kumar, V., and Steinbach, M. 2000. A comparison of document clustering techniques. In TextMining Workshop at KDD2000 (May 2000).

Kashima, Y., Hayase, Y., Yoshida, N., Manabe, Y., and Inoue, K. 2011. An investigation into the impact of software licenses on copy-and-paste reuse among OSS projects. In 2011 18th Working Conference on Reverse Engineering. IEEE, 28–32.

Kechagia, M., Spinellis, D., and Androutsellis-Theotokis, S. 2010. Open source licensing across package dependencies. In 2010 14th Panhellenic Conference on Informatics. IEEE, 27–32.

Keplinger, M. S. 1981. Computer Software–Its Nature and its Protection. Emory LJ 30, 483.

Kesan, J. P. and Gruner, R. S. 2020. Intellectual Property Compliance: Systematic Methods for Building and Using Intellectual Property. In Cambridge Handbook of Compliance, edited by D. Daniel Sokol & Benjamin van Rooij.

Kim, Y. and Stohr, E. A. 1998. Software reuse: survey and research directions. Journal of Management Information Systems 14 (4), 113–147.

Kitchenham, B. A., Budgen, D., and Brereton, P. 2015. Evidence-Based Software Engineering and Systematic Reviews. Chapman & Hall/CRC.

Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. 2018. Do developers update their library dependencies? Empirical Software Engineering 23 (1), 384–417.

Landes, W. M. and Posner, R. A. 1989. An economic analysis of copyright law. The Journal of Legal Studies 18 (2), 325–363.

Lanergan, R. G. and Grasso, C. A. 1984. Software engineering with reusable designs and code. IEEE Transactions on Software Engineering (5), 498–501.

Larman, C. and Basili, V. R. 2003. Iterative and incremental developments. a brief history. Computer 36 (6), 47–56.

Lau, B. and Ker, S. 2020. An Intellectual Property Law perspective on Open Source Software Licences - Taylor Vinters. ⟨URL: https://www.taylorvinters.com/article/an-intellectual-property-law-perspective-on-open-source-software-licences⟩.

Lee, D.-G. and Seo, Y.-S. 2018. A Study on the Identification of Open Source License Compatibility Violations. KIPS Transactions on Software and Data Engineering 7 (12), 451–460.

Lee, S., German, D. M., Hwang, S.-w., and Kim, S. 2015. Crowdsourcing Identification of License Violations. Journal of Computing Science and Engineering 9 (4), 190–203.

Lemley, M. A. 1994. Intellectual property and shrinkwrap licenses. S. Cal. L. Rev. 68, 1239.

Lerner, J. and Tirole, J. 2005. The scope of open source licensing. Journal of Law, Economics, and Organization 21 (1), 20–56.

Lindman, J., Rossi, M., Puustell, A., et al. 2011. Matching open source software licenses with corresponding business models. IEEE software.

Link, C. 2010. Patterns for the commercial use of open source: legal and licensing aspects. In Proceedings of the 15th European Conference on Pattern Languages of Programs, 1–10.

Liu, X., Huang, L., Ge, J., and Ng, V. 2019. Predicting licenses for changed source code. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 686–697.

Lokhman, A., Abdul-Rahman, S., Luoto, A., and Hammouda, I. 2011. Managing Open Source Legality Concerns–A Sustainability Catalyst. Proceedings of SOS 2011: Towards Sustainable Open Source, 13.

Lokhman, A., Luoto, A., Abdul-Rahman, S., and Hammouda, I. 2012. OSSLI: Architecture level management of open source software legality concerns. In IFIP International Conference on Open Source Systems. Springer, 356–361.

Manabe, Y., German, D. M., and Inoue, K. 2014. Analyzing the relationship between the license of packages and their files in free and open source software. In IFIP International Conference on Open Source Systems. Springer, 51–60.

Manabe, Y., Hayase, Y., and Inoue, K. 2010. Evolutional analysis of licenses in FOSS. In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), 83–87.

Marron, D. B. and Steel, D. G. 2000. Which countries protect intellectual property? The case of software piracy. Economic inquiry 38 (2), 159–174.

Mathur, A., Choudhary, H., Vashist, P., Thies, W., and Thilagam, S. 2012. An empirical study of license violations in open source projects. In 2012 35th Annual IEEE Software Engineering Workshop. IEEE, 168–176.

Mattmann, C. A., Oh, J.-H., Palsulich, T., McGibbney, L. J., Gil, Y., and Ratnakar, V. 2015. DRAT: An Unobtrusive, Scalable Approach to Large Scale Software License Analysis. In 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE, 97–101.

Meeker, H. 2017. Open source licensing: What every technologist should know. ⟨URL: https://opensource.org/licenses/category⟩.

Menell, P. S. 1989. An Analysis of the Scope of Copyright Protection for Application Programs. Stanford Law Review 41 (5), 1045. ISSN: 00389765. DOI: 10.2307/1228751.

Mertzel, N. J. 2008. Copying 0.03 percent of software code base not 'de minimis'. Journal of Intellectual Property Law & Practice 3 (9), 547–548.

Mikkonen, T. and Taivalsaari, A. 2019. Software reuse in the era of opportunistic design. IEEE Software 36 (3), 105–111.

Mlouki, O., Khomh, F., and Antoniol, G. 2016. On the detection of licenses violations in the android ecosystem. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Vol. 1. IEEE, 382–392.

Monden, A., Okahara, S., Manabe, Y., and Matsumoto, K. 2010. Guilty or not guilty: Using clone metrics to determine open source licensing violations. IEEE software 28 (2), 42–47.

Moreau, B., Serrano-Alvarado, P., and Desmontils, E. 2018. CaLi: A Lattice-Based Model for License Classifications.

Moreau, B., Serrano-Alvarado, P., Perrin, M., and Desmontils, E. 2019. Modelling the compatibility of licenses. In European Semantic Web Conference. Springer, 255–269.

Morszczyzna, M. 2017. What's really wrong with node_modules and why this is your fault. ⟨URL: https://hackernoon.com/whats-really-wrong-with-node-modules-and-why-this-is-your-fault-8ac9fa893823?gi=8cf0367f74dc⟩.

Mozilla 2020. MPL 2.0 FAQ. ⟨URL: https://www.mozilla.org/en-US/MPL/2.0/FAQ/⟩.

Nejad, N. M., Scerri, S., Auer, S., and Sibarani, E. M. 2016. Eulaide: Interpretation of end-user license agreements using ontology-based information extraction. In Proceedings of the 12th International Conference on Semantic Systems, 73–80.

nexB inc. 2020. Licenses. ⟨URL: https://enterprise.dejacode.com/licenses/⟩.

O'Hare, M. 1982. Copyright and the protection of economic rights. Journal of Cultural Economics 6 (1), 33–48.

Open Source Initiative 2020a. Open Source Licenses by Category. ⟨URL: https://opensource.org/licenses/category⟩.

Open Source Initiative 2020b. The 2-Clause BSD License. ⟨URL: https://opensource.org/licenses/BSD-2-Clause⟩.

Open Source Initiative 2020c. The 3-Clause BSD License. ⟨URL: https://opensource.org/licenses/BSD-3-Clause⟩.

Open Source Initiative 2020d. The Open Source Definition. ⟨URL: https://opensource.org/osd⟩.

Papazafeiropoulou, A. and Spanaki, K. 2016. Understanding governance, risk and compliance information systems (GRC IS): The experts view. Information Systems Frontiers 18 (6), 1251–1263.

Paschalides, D. and Kapitsaki, G. M. 2016. Validate your SPDX files for open source license violations. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 1047–1051.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. A design science research methodology for information systems research. Journal of management information systems 24 (3), 45–77.

Pellegrini, T., Havur, G., Steyskal, S., Panasiuk, O., Fensel, A., Mireles, V., Thurner, T., Polleres, A., Kirrane, S., and Schönhofer, A. 2019. DALICC: A License Management Framework for Digital Assets. Proceedings of the Internationales Rechtsinformatik Symposion (IRIS) 10.

Peterson, C. 2018. How I coined the term 'open source'. ⟨URL: https://opensource.com/article/18/2/coining-term-open-source-software⟩.

Phipps, S. 2013. Open source needs to clean up its language. ⟨URL: https://www.infoworld.com/article/2612747/open-source-needs-to-clean-up-its-language.html⟩.

Pina, P. 2011. Computer Games and Intellectual Property Law: Derivative Works, Copyright and Copyleft. In Business, Technological, and Social Dimensions of Computer Games: Multidisciplinary Developments. IGI Global, 464–475.

PRISMA 2015. PRISMA. Transparent reporting of systematic reviews and meta-analyses. ⟨URL: http://www.prisma-statement.org/⟩.

Ragkhitwetsagul, C. and Krinke, J. 2019. Siamese: scalable and incremental code clone search via multiple code representations. Empirical Software Engineering 24 (4), 2236–2284.

Ragkhitwetsagul, C., Krinke, J., and Oliveto, R. 2018. Awareness and experience of developers to outdated and license-violating code on stack overflow: An online survey. arXiv preprint arXiv:1806.08149.

Riehle, D. and Harutyunyan, N. 2019. Open-source license compliance in software supply chains. In Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability. Springer, 83–95.

Rigamonti, C. P. 2006. Deconstructing moral rights. Harv. Int'l LJ 47, 353.

Romansky, S., Chen, C., Malhotra, B., and Hindle, A. 2018. Sourcerer's Apprentice and the study of code snippet migration. arXiv preprint arXiv:1808.00106.

Rosen, L. 2001. The unreasonable fear of infection. Open Magazine.

Rosen, L. 2005. Open source licensing. Vol. 692. Prentice Hall.

RoyChowdhury, S., Gangadharan, G., Silveira, P., and D'Andrea, V. 2010. From ODRL-S to low-Level DSL: a case study based on license compliance in service oriented systems. In Proceedings of the 8th international workshop for technical, economic and legal aspects of business models for virtual goods.

Ruffin, C. and Ebert, C. 2004. Using open source software in product development: A primer. IEEE software 21 (1), 82–86.

Ruskin, J. 2018. Improving your OSS dependency workflow with Licensed. ⟨URL: https://github.blog/2018-03-07-improving-your-oss-dependency-workflow-with-licensed/⟩.

Ryan, P. 2009. Cisco settles FSF GPL lawsuit, appoints compliance officer. ⟨URL: https://arstechnica.com/information-technology/2009/05/cisco-settles-fsf-gpl-lawsuit-appoints-compliance-officer/⟩.

Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K., and Lopes, C. V. 2016. SourcererCC: Scaling code clone detection to big-code. In Proceedings of the 38th International Conference on Software Engineering, 1157–1168.

Sass, R. 2015. Top 10 Common Development and Distribution License (CDDL) Questions Answered. ⟨URL: https://resources.whitesourcesoftware.com/blog-whitesource/top-10-cddl-questions-answered⟩.

Schoettle, H. 2019. Open Source License Compliance-Why and How? Computer 52 (8), 63–67.

Sen, R., Subramaniam, C., and Nelson, M. L. 2011. Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? Decision support systems 52 (1), 199–206.

Sen, R., Subramaniam, C., and Nelson, M. L. 2008. Determinants of the choice of open source software license. Journal of Management Information Systems 25 (3), 207–240. ISSN: 07421222. DOI: 10.2753/MIS0742-1222250306.

Singi, K., Kaulgud, V., Bose, R. J. C., and Podder, S. 2019. CAG: compliance adherence and governance in software delivery using blockchain. In 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 32–39.

Sonnenberg, C. and Vom Brocke, J. 2012. Evaluations in the science of the artificial–reconsidering the build-evaluate pattern in design science research. In International Conference on Design Science Research in Information Systems. Springer, 381–397.

SPDX Workgroup 2018. License Exceptions. ⟨URL: https://spdx.org/licenses/exceptions-index.html⟩.

SPDX Workgroup 2020a. SPDX. ⟨URL: https://spdx.dev/⟩.

SPDX Workgroup 2020b. SPDX Specification submitted to ISO. ⟨URL: https://spdx.dev/spdx-specification-submitted-to-iso/⟩.

Stack Exchange Inc. 2020. Public Network Terms of Service. ⟨URL: https://stackoverflow.com/legal/terms-of-service/public#licensing⟩.

Stallman, R. 1985. The GNU Manifesto. ⟨URL: https://www.gnu.org/gnu/manifesto.en.html⟩.

Stoltz, M. L. 2005. The penguin paradox: How the scope of derivative works in copyright affects the effectiveness of the GNU GPL. BUL Rev. 85, 1439.

Synopsis 2018. Synopsis: Open Source Security and Risk Analysis. Network Security 2018 (6), 3. ISSN: 1353-4858. DOI: https://doi.org/10.1016/S1353-4858(18)30051-5.

Tamrawi, A., Nguyen, H. A., Nguyen, H. V., and Nguyen, T. N. 2012. Build code analysis with symbolic evaluation. In 2012 34th International Conference on Software Engineering (ICSE). IEEE, 650–660.

Tang, W., Chen, D., and Luo, P. 2018. Bcfinder: A lightweight and platform-independent tool to find third-party components in binaries. In 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 288–297.

Tuunanen, T., Koskinen, J., and Kärkkäinen, T. 2006a. Asla: reverse engineering approach for software license information retrieval. In Conference on Software Maintenance and Reengineering (CSMR'06). IEEE, 4–pp.

Tuunanen, T., Koskinen, J., and Kärkkäinen, T. 2006b. Retrieving open source software licenses. In IFIP International Conference on Open Source Systems. Springer, 35–46.

Tuunanen, T., Koskinen, J., and Kärkkäinen, T. 2009. Automated software license analysis. Automated Software Engineering 16 (3-4), 455–490.

United Nations 2020. Universal Declaration of Human Rights. ⟨URL: https://www.un.org/en/universal-declaration-human-rights/⟩.

US Commission on New Technological Uses of Copyrighted Works (CONTU) 1978. Final report of the National Commission on New Technological Uses of Copyrighted Works. US Congress.

Välimäki, M. and Oksanen, V. 2005. How to Manage IPR Infringement Risks in Open Source Development. Intellectual Property Beyond Rights", IPR University Center, WSOY, Finland, 347–364.

Van Der Burg, S., Dolstra, E., McIntosh, S., Davies, J., German, D. M., and Hemel, A. 2014. Tracing software build processes to uncover license compliance inconsistencies. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 731–742.

Vaughan-Nichols, S. J. 2015. VMware sued for failure to comply with Linux license. ⟨URL: https://www.zdnet.com/article/vmware-sued-for-failure-to-comply-with-linuxs-license/⟩.

Venable, J. 2010. Design science research post hevner et al.: criteria, standards, guidelines, and expectations. In International Conference on Design Science Research in Information Systems. Springer, 109–123.

Venable, J., Pries-Heje, J., and Baskerville, R. 2016. FEDS: a framework for evaluation in design science research. European journal of information systems 25 (1), 77–89.

Vendome, C. 2015. A large scale study of license usage on GitHub. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 2. IEEE, 772–774.

Vendome, C. 2016. Assisting developers with license compliance. In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). IEEE, 811–814.

Vendome, C., Bavota, G., Di Penta, M., Linares-Vásquez, M., German, D. M., and Poshyvanyk, D. 2017a. License usage and changes: a large-scale study on gitHub. Empirical Software Engineering 22 (3), 1537–1577.

Vendome, C., German, D. M., Di Penta, M., Bavota, G., Linares-Vásquez, M., and Poshyvanyk, D. 2018. To Distribute or Not to Distribute? Why Licensing Bugs Matter. In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 268–279.

Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., German, D. M., and Poshyvanyk, D. 2015. License usage and changes: a large-scale study of java projects on github. In 2015 IEEE 23rd International Conference on Program Comprehension. IEEE, 218–228.

Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., German, D. M., and Poshyvanyk, D. 2017b. Machine learning-based detection of open source license exceptions. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 118–129.

Viseur, R. 2016. A FLOSS License-selection Methodology for Cloud Computing Projects. In CLOSER (1), 129–136.

Vom Brocke, J., Hevner, A. R., and Maedche, A. 2020. Introduction to Design Science Research. Springer International Publishing, 1–13. ISBN: 9783030467814. DOI: 10.1007/978-3-030-46781-4_1. ⟨URL: http://dx.doi.org/10.1007/978-3-030-46781-4_1⟩.

Von Krogh, G. and Von Hippel, E. 2003. Special issue on open source software development.

Weckert, J. 1997. Intellectual property rights and computer software. Business Ethics: A European Review 6 (2), 101–109.

Weissman, N. 2016. I Didn't Believe It Either. Microsoft No.1 for Open Source! ⟨URL: https://resources.whitesourcesoftware.com/blog-whitesource/i-didn-t-believe-it-either-microsoft-no-1-for-open-source⟩.

Wikipedia 2020. Threshold of originality. ⟨URL: https://en.wikipedia.org/wiki/Threshold_of_originality⟩.

Winslow, S. 2020. Copyright Notices in Open Source Software Projects. ⟨URL: https://www.linuxfoundation.org/blog/2020/01/copyright-notices-in-open-source-software-projects/⟩.

WIPO 2016. Understanding Copyright and related rights. WIPO Publication, 1–26. ISSN: 03361500. DOI: 10.3406/colan.1975.4181. ⟨URL: https://www.wipo.int/edocs/pubdocs/en/wipo_pub_909_2016.pdf⟩.

WIPO 2020a. Copyright Protection of Computer Software. ⟨URL: https://www.wipo.int/copyright/en/activities/software.html⟩.

WIPO 2020b. What is Intellectual Property? ⟨URL: https://www.wipo.int/edocs/pubdocs/en/wipo_pub_450_2020.pdf⟩.

WTO 2020. What are intellectual property rights? ⟨URL: https://www.wto.org/english/tratop_e/trips_e/intel1_e.htm⟩.

Wu, Y., Manabe, Y., German, D. M., and Inoue, K. 2017a. How are Developers Treating License Inconsistency Issues? A Case Study on License Inconsistency Evolution in FOSS Projects. In IFIP International Conference on Open Source Systems. Springer, 69–79.

Wu, Y., Manabe, Y., Kanda, T., German, D. M., and Inoue, K. 2015. A method to detect license inconsistencies in large-scale open source projects. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, 324–333.

Wu, Y., Manabe, Y., Kanda, T., German, D. M., and Inoue, K. 2017b. Analysis of license inconsistency in large collections of open source projects. Empirical Software Engineering 22 (3), 1194–1222.

Wu, Y., Wang, S., Bezemer, C.-P., and Inoue, K. 2019. How do developers utilize source code from stack overflow? Empirical Software Engineering 24 (2), 637–673.

Xu, H., Yang, H., Wan, D., and Wan, J. 2010. The design and implement of open source license tracking system. In 2010 International Conference on Computational Intelligence and Software Engineering. IEEE, 1–4.

Yun, H. Y., Joe, Y. J., Jung, B.-O., and Shin, D. 2017a. Method for License Compliance of Open Source Software. In Proceedings of the Korea Information Processing Society Conference. Korea Information Processing Society, 548–550.

Yun, H. Y., Joe, Y. J., and Shin, D. M. 2017b. Method of license compliance of open source software governance. In 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS). IEEE, 83–86.

Zhang, H., Shi, B., and Zhang, L. 2010. Automatic checking of license compliance. In 2010 IEEE International Conference on Software Maintenance. IEEE, 1–3.

APPENDIX 1 CATEGORIZATION OF OSI-APPROVED LICENSES

TABLE 13 Categorization of OSI-approved licenses.

Permissive	Weak copyleft	Strong copyleft
0-clause BSD License (0BSD)	Adaptive Public License (APL-1.0)	CeCILL License 2.1 (CECILL-2.1)
1-clause BSD License (BSD-1-Clause)	Apple Public Source License (APSL-2.0)	Cryptographic Autonomy License v.1.0 (CAL-1.0)
2-clause BSD License (BSD-2-Clause)	Common Public Attribution License 1.0 (CPAL-1.0)	European Union Public License 1.2 (EUPL-1.2)
3-clause BSD License (BSD-3-Clause)	Common Development and Distribution License 1.0 (CDDL-1.0)	Frameworx License (Frameworx-1.0)
Academic Free License 3.0 (AFL-3.0)	Eclipse Public License 2.0 (EPL-2.0)	GNU Affero General Public License version 3 (AGPL-3.0)
Apache License 2.0 (Apache 2.0)	GNU Lesser General Public License version 2.1 (LGPL-2.1)	GNU General Public License version 2 (GPL-2.0)
Artistic License 2.0 (Artistic-2.0)	GNU Lesser General Public License version 3 (LGPL-3.0)	GNU General Public License version 3 (GPL-3.0)
Attribution Assurance License (AAL)	IBM Public License 1.0 (IPL-1.0)	Licence Libre du Québec – Réciprocité forte (LiLiQ-R+) version 1.1 (LiliQ-R+)
Boost Software License (BSL-1.0)	Licence Libre du Québec – Réciprocité (LiLiQ-R) version 1.1 (LiliQ-R)	Nethack General Public License (NGPL)
BSD-3-Clause-LBNL	Microsoft Reciprocal License (MS-RL)	Non-Profit Open Software License 3.0 (NPOSL-3.0)
BSD+Patent (BSD-2-Clause-Patent)	Motosoto License (Motosoto)	OCLC Research Public License 2.0 (OCLC-2.0)
Computer Associates Trusted Open Source License 1.1 (CATOSL-1.1)	Mozilla Public License 2.0 (MPL-2.0)	Open Software License 3.0 (OSL-3.0)
eCos License version 2.0 (eCos-2.0)	NASA Open Source Agreement 1.3 (NASA-1.3)	RealNetworks Public Source License V1.0 (RPSL-1.0)
Educational Community License, Version 2.0 (ECL-2.0)	Nokia Open Source License (Nokia)	Reciprocal Public License 1.5 (RPL-1.5)
Eiffel Forum License V2.0 (EFL-2.0)	OSET Public License version 2.1	Simple Public License 2.0 (SimPL-2.0)
Entessa Public License (Entessa)	Q Public License (QPL-1.0)	Sleepycat License (Sleepycat)
EU DataGrid Software License (EUDatagrid)	Ricoh Source Code Public License (RSCPL)	Sybase Open Watcom Public License 1.0 (Watcom-1.0)
Fair License (Fair)	Sun Public License 1.0 (SPL-1.0)	
Historical Permission Notice and Disclaimer (HPND)	Upstream Compatibility License v1.0	
IPA Font License (IPA)	wxWindows Library License (WXwindows)	
ISC License (ISC)		
LaTeX Project Public License 1.3c (LPPL-1.3c)		
Lawrence Berkeley National Labs BSD Variant License (BSD-3-Clause-LBNL)		
Licence Libre du Québec – Permissive (LiLiQ-P) version 1.1 (LiliQ-P)		
Lucent Public License Version 1.02 (LPL-1.02)		
Microsoft Public License (MS-PL)		
MirOS Licence (MirOS)		
MIT License (MIT)		
Mulan Permissive Software License v2 (MulanPSL - 2.0)		
Multics License (Multics)		
Naumen Public License (Naumen)		
NTP License (NTP)		
Open Group Test Suite License (OGTSL)		
OpenLDAP Public License Version 2.8 (OLDAP-2.8)		
PHP License 3.01 (PHP-3.01)		
The PostgreSQL License (PostgreSQL)		
Python License (Python-2.0)		
CNRI Python license (CNRI-Python)		
SIL Open Font License 1.1 (OFL-1.1)		
Universal Permissive License (UPL)		
University of Illinois/NCSA Open Source License (NCSA)		
Unicode Data Files and Software License		
The Unlicense		
Vovida Software License v. 1.0 (VSL-1.0)		
W3C License (W3C)		
X.Net License (Xnet)		
Zero-Clause BSD / Free Public License 1.0.0 (0BSD)		
Zope Public License 2.0 (ZPL-2.0)		
zlib/libpng license (Zlib)		

APPENDIX 2 EXCLUDED PUBLICATIONS IN PHASE 3 OF SELECTION PROCESS

TABLE 14 Excluded publications in phase 3 of selection process.

Authors	Exclusion rationale
An et al. (2017)	No automated tools or methods for OSS license compliance described. Uses method described in Mlouki et al. (2016).
Azhakesan and Paulisch (2020)	No automated tools or methods for OSS license compliance described. Only listed for upcoming talk.
Boughanmi (2010)	No automated tools or methods for OSS license compliance described.
Duan et al. (2017)	Related to clone detection of binaries, which is not the preferred format for reuse.
Fendt and Jaeger (2019)	No automated tools or methods for OSS license compliance described.
Feng et al. (2019)	Not related to OSS license compliance in reuse. Violation detection instead.
Fujita and Tsukada (2012)	Not related to OSS licenses.
Germonprez et al. (2012)	No automated tools or methods for OSS license compliance described.
Gordon (2014)	Tool demonstration. Quality of research raises questions.
Harutyunyan et al. (2019)	No automated tools or methods for OSS license compliance described. However, important background info.
Harutyunyan and Riehle (2019b)	No automated tools or methods for OSS license compliance described. Instead focuses on OSS governance.
Harutyunyan and Riehle (2019a)	No automated tools or methods for OSS license compliance described. Focuses on OSS governance and compliance practices.
Heirendt et al. (2017)	Tool demonstration. Quality of research raises questions.
Hemel et al. (2011)	Not related to OSS license compliance in reuse. Violation detection instead.
Ishio et al. (2017)	No automated tools or methods for OSS license compliance described.
Kesan and Gruner (2020)	Not related to OSS.
Lindman et al. (2011)	Related to license selection, not compliance.
Link (2010)	No automated tools or methods for OSS license compliance described.
Lokhman et al. (2011)	No automated tools or methods for OSS license compliance described.

Manabe et al. (2010)	No automated tools or methods for OSS license compliance described. Only uses Ninka.
Monden et al. (2010)	Clone detection metrics exploration. Not related to license compliance.
Moreau et al. (2018)	Working paper. Results presented in Moreau et al. (2019).
Papazafeiropoulou and Spanaki (2016)	No automated tools or methods for OSS license compliance described.
Riehle and Harutyunyan (2019)	No automated tools or methods for OSS license compliance described.
Romansky et al. (2018)	Quality of research raises questions. Published in non peer reviewed site.
RoyChowdhury et al. (2010)	Not related to OSS reuse.
Schoettle (2019)	No automated tools or methods for OSS license compliance described. However, important background info.
Tamrawi et al. (2012)	Not related to OSS license compliance in reuse. Improvement to ASLA dependency map creation.
Tang et al. (2018)	Not related to OSS license compliance in reuse. Violation detection instead.
Vendome (2015)	No automated tools or methods for OSS license compliance described.
Vendome et al. (2015)	No automated tools or methods for OSS license compliance described.
Vendome (2016)	No automated tools or methods for OSS license compliance described.
Vendome et al. (2017a)	No automated tools or methods for OSS license compliance described.
Viseur (2016)	Related to license selection, not compliance.
Wu et al. (2017b)	No automated tools or methods for OSS license compliance described. Uses method described in Wu et al. (2015)
Wu et al. (2017a)	No automated tools or methods for OSS license compliance described. Uses method described in Wu et al. (2015).
Wu et al. (2019)	No automated tools or methods for OSS license compliance described.