Assisting Software Developers With License Compliance

Christopher Vendome
College of William and Mary - Arts & Sciences, cgvendome@email.wm.edu

Assisting Software Developers with License Compliance

Christopher Vendome

Williamsburg, Virginia

Bachelor of Science, Emory University, 2012
Master of Science, College of Williams & Mary, 2014

A Dissertation presented to the Graduate Faculty
of The College of William & Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

College of William & Mary
August 2018

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Christopher Vendome

Christopher Vendome

Approved by the Committee, May 2018

[illegible]

Committee Chair
Associate Professor Denys Poshyvanyk, Computer Science
College of William & Mary

[illegible]

Associate Professor Peter Kemper, Computer Science
College of William & Mary

[illegible]

Professor Andreas Stathopoulos, Computer Science
College of William & Mary

[illegible]

Assistant Professor Xu Liu, Computer Science
College of William & Mary

Robert Michael Lewis, FOR DANIEL GERMAN

Professor Daniel Germán, Computer Science
University of Victoria, Canada

ABSTRACT

Open source licensing determines how open source systems are reused, distributed, and modified from a legal perspective. While it facilitates rapid development, it can present difficulty for developers in understanding due to the legal language of these licenses. Because of misunderstandings, systems can incorporate licensed code in a way that violates the terms of the license. Such incompatibilities between licensing can result in the inability to reuse a particular library without either relicensing the system or redesigning the architecture of the system. Prior efforts have predominantly focused on license identification or understanding the underlying phenomena without reasoning about compatibility in a broad scale.

The work in this dissertation first investigates the rationale of developers and identifies the areas that developers struggle with respect to free/open source software licensing. First, we investigate the diffusion of licenses and the prevalence of license changes in a large scale empirical study of 16,221 Java systems. We observed a clear lack of traceability and a lack of standardized licensing that led to difficulties and confusion for developers trying to reuse source code. We further investigated the difficulty by surveying the developers of the systems with license changes to understand why they first adopted a license and then changed licenses. Additionally, we performed an analysis on issue trackers and legal mailing lists to extract licensing bugs. From these works, we identified key areas in which developers struggled and needed support.

While developers need support to identify license incompatibilities and understand both the cause and implications of the incompatibilities, we observed that state-of-the-art license identification tools did not identify license exceptions. Since these exceptions directly modify the license terms (either the permissions granted by the license or the restrictions imposed by the license), we proposed an approach to complement current license identification techniques in order to classify license exceptions. The approach relies on supervised machine learners to classify the licensing text to identify the particular license exceptions or the lack of a license exception.

Subsequently, we built an infrastructure to assist developers with evaluating license compliance warnings for their system. The infrastructure evaluates compliance across the dependency tree of a system to ensure it is compliant with all of the licenses of the dependencies. When an incompatibility is present, it notes the specific library/libraries and the conflicting license(s) so that the developers can investigate these compliance warnings, which would prevent distribution of their software, in their system. We conduct a study on 121,094 open source projects spanning 6 programming languages, and we demonstrate that the infrastructure is able to identify license incompatibilities between these projects and their dependencies.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

During the course of this dissertation, I was fortunate to have the honor of working with great people and outstanding researchers from several countries, including the USA, Italy, Canada, and Colombia. First, I would like to thank my advisor Denys Poshyvanyk for all of his mentorship and guidance. My thanks also go to my research collaborators, Massimiliano Di Penta, Daniel Germán, and Gabriele Bavota, for all of their exceptional feedback and advice that has helped me grow to be a better researcher. I would also like to thank Mario Linares-Vásquez, researcher collaborator and close friend. Additionally, I would like to thank my parents and my sister for all of their support and encouragement throughout my entire academic career.

LIST OF TABLES

LIST OF FIGURES

Assisting Software Developers with License Compliance

Chapter 1

Introduction

Software reuse allows for rapid development by leveraging existing components. These components can be modified (i.e. derivative work) or incorporated into another system. To protect the original authors, the code contains a license to describe under which conditions the code can be modified, distributed, and utilized. However, software licenses each contain different restrictions, which introduce the potential for incompatibilities. Without a license, software would be subject to copyright laws, which would inhibit modifying and redistributing the software. The license limits the scope of copyright under certain conditions specified in the text of the license. In this dissertation, we focus on free/open source licenses and how to support developer with the challenges that they face when reusing or distributing free/open source software.

In this work, the term "free/open" refers software licenses under a free license and an open source license. In order for a license to be considered *free*, it must satisfy the criteria specified by the Free Software Foundation [44]. It is important to note that "free" is not a matter of financial cost, but "is a matter of liberty" [44]. Similarly, open source licenses meet the open source definition as defined by the Open Source Initiative [26]. In general, these two definitions have significant overlap and almost all open source licenses also satisfy the free software license criteria. It is the usage of these free/open source licenses that determines whether a piece of software is free/open source, and not just whether the source code is available. For example, the source code may be available online in GitHub, but it is still closed source if it does not include a license.

Although two licenses may be free/open source licenses, it does not guarantee that they are compatible with each other. There are two type of free/open source licenses: permissive and restrictive. A key difference between these types of licenses are that restrictive licenses require that any software using and distributing that code or binary to have the same license (*e.g., GPL-2.0*). Otherwise, a developer would not be able to legally distribute the software. Additionally, compliance can be made more difficult, because the direction of reuse matter. By direction, software licensed with *GPL-3.0* can contain depend on software licensed with *Apache-2.0*, but the opposite (*i.e., Apache-2.0* software depending on *GPL-3.0* software) would not be allowed.

In addition to compliance, there are different ways of documenting the licensing of software and the lack of standardization can also cause difficulty. For example, developers can include the license text in the header of the source code of each file, but they may not have a license file in the root directory. Alternatively, one project may place the license in a README file, while another has its own LICENSE file. Furthermore, there can be mismatches or inconsistencies between the licenses in the source code and the license files. While there are initiatives, like SPDX, that are trying to address some of the problems, there is limited support for developers with respect to automatically identifying potential licensing issues that would prevent the distribution of their software.

Previous work has investigated license identification and the current state-of-the-art has 96% precision. The approach was developed by German *et al.* [106] and utilizes pattern-matching and critical terms to identify licenses. Other tools also exist [107] [146], but Ninka currently is considered the most precise tool.

The problem of identifying the license of a binary has also been investigated by Di Penta *et al.* [99], by utilizing Google Code Search to infer the license of byte-code. While the approach presented promising results, Google Code Search no longer exists, and so the approach is not currently feasible. However, it motivates the need to trace byte-code to source code in order to extract licensing. While Davies *et al.* [98] focus on identifying the version of java archives, their approach establishes provenance from byte-code by using anchored class signatures. They addressed a key drawback of existing clone detection - scalability.

Di Penta *et al.* [100] analyzed the revision history of six open source systems and extracted license migration patterns. While the authors were unable to classify the migrations into generalizable patterns, they were able to empirically demonstrate that the systems experience changes in license type and license version. German *et al.* [104] explored cloned code fragments between Linux, and FreeBSD and OpenBSD to investigate the licensing implications of the code fragment migrations. The authors observed that the migrations occurred most prevalently in the movement toward a more restrictive license. Manabe *et al.* [119] investigated license changes among FreeBSD, OpenBSD, Eclipse, and ArgoUML and concluded that each system had a different change pattern. This research demonstrates that licensing changes exist in large system and that while there are patterns, they might be application specific.

The analysis of license incompatibilities has been briefly investigated in prior work. German *et al.* [105] investigated how incompatibilities are handled by developers. They constructed a model with the advantages and disadvantages of specific change patterns to suggest when a license change is appropriate. German *et al.* [103] also investigated license incompatibilities in the packages of Fedora Linux. They explored the consistency between declared package licenses and the actual source licenses, and they were able to identify confirmed incompatibilities between packages used as dependencies. These results indicate that license compatibility is a problem in open source development and there is a need for tools to support developers with license compliance.

Since the licensing of a project may change during evolution and maintenance, it is important to understand the reason for these changes. Combining the results of the previous work, there can be hypothesized reasons for the changes, but it lacks empirical evidence to make any sound claims. While reuse and dependencies are likely to impact license changes, it is important to understand in what way and to what extent. Our preliminary work aimed at answering these questions and analyzing changes in licensing on a much larger dataset (16,221 Java systems) [153] [152]. Prior work proves the presence of incompatibilities, especially due to dependencies. However, it relied upon explicit dependency information as opposed to extracting the dependencies.

Our research will aid developers by automating the analysis of their system. The work in this dissertation focuses on assisting developers with free/open source licenses and does not consider closed source licenses. Our goal is two-fold: first, we want to automatically audit the system and guarantee license compliance; second, we want to support developers in fixing license incompatibilities. The approach needs to identify the correct licensing of the source as well as understand the way in which the code is being utilized (*e.g.,* dependency or derivative work). Subsequently, licenses must be analyzed in conjunction with their use case to determine the validity of a violation. These license violations would be presented to the developer listing the component and the conflicting license. These steps allow developers to understand where and why a component is incompatible.

In summary, this dissertation makes the following research contributions:

- **Empirical Studies.** The first part of the dissertation was a set of empirical studies to uunderstand phenomena related to license usage and license changes as well as the difficult that developers experience with licensing. We identified prevalence of licenses and the migration patterns on a large set of Java systems. Additionally, the work surveyed practitioners to understanding the practitioners' perspective, which has widely been ignored from prior studies. We utilized approaches for large scale mining of software repositories as well as qualitative analysis based on formal grounded theory [95] principles. These studies show that developers experience a diverse set of problems with licensing. Two key insights were a general lack of understandability of the implications of various licenses (even simpler licenses) as well as the impact of external factors, such as commercial partners.

- **Catalog of Licensing Bugs.** From the empirical studies, it is apparent that developers experience many problems with licensing and struggle understanding the litigious nature of these licenses. We investigated issue trackers and legal mailing lists to understand these problems that may prevent distributing software or lead to difficulty for developers when understanding software licensing. We performed a formal open coding that resulted in a catalog of 7 categories with 21 sub-categories. This work demonstrates that developers lack the necessary support when trying to understanding software licensing and when trying to determine if they are reusing software properly.

- **Improving License Identification.** While current state-of-the-art tools, like ninka [106], have achieved a high accuracy for license identification, these existing approaches were not designed to identify the presence of a license exception. We designed a machine learning based approach to classify license exceptions or the lack of exception (i.e., standard license) in order more accurately provide compliance analysis. Additionally, our preliminary analysis of license exceptions that utilized a broad heuristic search for license exception was able to identify new license exceptions that had not been previously listed among the known exceptions.

- **License Compliance Infrastructure.** To assist developers with software licensing, we built an infrastructure to analyze license compatibility of a system. The approach relies upon a compatibility matrix, which consists of pair-wise analysis of compatibility between licenses, to generate warnings of potential license incompatibilities. The approach further analyzes the system to extract all of the dependencies along the dependency tree. Such an analysis is critical for a compliance engine, since a project needs to comply with all of the licenses of the reused software. Thus, it is important to ensure that a project's dependencies do not have a potential incompatibility as well, since the system would inherit those potential incompatibilities. It then presents the developer a report of warnings with the components that are responsible for the potential incompatibility as well as their licenses so that the developer can investigate further and remedy the potential licensing issues.

Chapter 2

Empirical Studies

In this chapter, we present a set of empirical studies conducted to understand phenomena related to software licensing within evolving systems. We first are interested in the diffusion of free/open source licenses within a large dataset of Java projects. Additionally, we investigate the extent to which these licenses change. While we can glean insights from the empirical observations, the rationale behind choosing a particular license or changing licenses can be influenced by many external factors. For example, the Apache Foundation asserts that all contributions must be licensed under the *Apache-2.0* license (although they do allow some exceptions for "non-essential" code). Additionally, developers may design their software system with the goal of commercial integration, leading to a more permissive license (*e.g., MIT* license).

The empirical studies aim to identify the difficulty that developers face with respect to free/open source licensing in order to guide researchers interested in assisting developers with licensing issues. The empirical results demonstrate a lack of traceability of license changes that could directly impact developers reusing existing code within their system. Similarly, we identify that there is an inherent bias that developers have towards using particular licenses. While it may be easy for a developer to relicense their own source code in order to utilize an external dependency, this recommendation may conflict with a personal bias and the developer may opt to change dependencies to achieve compliance.

Finally, we observe that the litigious nature of the text of licenses are difficult for developers. It is important for tools to contextualize license incompatibilities in such a way that developers understand how they violated the terms of the license(s).

2.1 Free/Open Source Software License Usage and License Changes

When developers (or organizations) decide to make a project available as open source, they can license their code under one or many different existing licenses. The choice may be dictated by the set of dependencies that the project has (*e.g.,* what libraries it uses), since those dependencies might have specific licensing constraints to those that reuse them. For instance, if a project links (statically) some *GPL* code, then it must be released under the same *GPL* version; failing to fulfill such a constraint could create a potential legal risk. Also, as shown by Di Penta *et al.* [100], the choice of the licenses in a free/open source project may have a massive impact on its success, as well as on projects using it. For example—as it happened for the IPFilter project [29]—a highly restrictive license may prevent others from redistributing the project (in the case of IPFilter, this caused its exclusion from the OpenBSD distributions). An opposite case is the one of MySQL connect drivers, originally released under *GPL-2.0*, whose license was modified with an exception [127] to allow the driver's inclusion in other software released under some open source licenses, which would otherwise be incompatible with the *GPL* (*e.g.,* the original *Apache* license). In summary, the choice of the license—or even a decision to change an existing license—is a crucial crossroad point in the context of software evolution of every free/open source project.

In order to encourage developers to think about licensing issues early in the development process, some forges (*e.g.,* GitHub) have introduced mechanisms such as the possibility of picking the project license at the time the repository is created. Also, there are some web sites (*e.g.,* `http://choosealicense.com`) helping developers to choose a license. Furthermore, there are numerous research efforts aimed at supporting developers in classifying source code licenses [107, 106] and identifying licensing incompatibilities [103]. Even initiatives such as the Software Package Data Exchange (SPDX) [36] have been aimed at proposing a formal model to document the license of a system. However, despite of the effort put by the open source community, researchers, and independent companies, it turns out that developers usually do not have a clear idea on the exact consequences of licensing (or not) their code using a specific license, or they are unsure (for example, on how to re-distribute code licensed with a dual license among the other issues [153]).

This work reports the results of a large empirical study aimed at quantitatively and qualitatively investigating when and why licenses change in open source projects, and to what extent is it possible to establish traceability links between licensing related-discussions and changes. First, we perform a quantitative analysis conducted on 16,221 Java projects hosted on GitHub. To conduct this study, we first mined the entire change history of the projects, extracting the license name (*e.g.,GPL* or *Apache*) and version (*e.g.,v1*, *v2*), when applicable, from each of the 4,665,611 files involved in a total of 1,731,828 commits. Starting from this data, we provide quantitative evidence on (i) the diffusion of licenses in free/open source systems, (ii) the most common license-change patterns, and (iii) the traceability between the license changes to both the commit messages and the issue tracker discussions. After that, following an open coding approach inspired by grounded theory [95], we qualitatively analyze a sample of commit messages and issue tracker discussions likely related to license changes. Such a qualitative analysis has been performed on 1,160 projects written in seven different languages: 159 C, 91 C++, 78 C#, 324 Java, 166 Javascript, 147 Python, and 195 Ruby projects. The results of this analysis provide a rationale on why developers adopt specific license(s), both for initial licensing and for licensing changes.

Our findings below suggest that determining the appropriate license of a software project is far from trivial and that a community's usage and expectations can influence developers when picking a license. We also observe that licensing expectations may be different based on the programming language. Although choosing a license is considered important for developers, even from early releases of their projects, forges and third party-tools provide little or no support to developers when performing licensing-related tasks, *e.g.,* picking a license, declaring the license of a project, changing license from a restrictive one towards a more permissive one (or *vice versa*) and, importantly, keeping track of the rationale for license changes.

Complete details of the design of the study can be found in our ICPC'15 and EMSE'17 publication [152, 149]

2.1.1 RQ$_1$: What is the usage of different licenses in Java Systems from GitHub?

Using the MARKOS code analyzer [90], we extracted the licensing of each file at commit-level for each project. Fig. 2.1 depicts the percentage of licenses that were first introduced into a project in the given year, which we refer to as *relative license usage*. We only report the first occurrence of each license committed to any file of the project. To ease readability, the bars are grouped by permissive (dashed bars) or restrictive licenses (solid bars). Additionally, we omit data prior to 2002 due to the limited number of projects created during those years in our sampled dataset.

For the year 2002, we observed that restrictive licenses and permissive licenses had been used approximately equally with a slight bias towards using restrictive licenses. Although the *LGPL-2.1* and *LGPL-2.1+* variants are restrictive licenses, they are less restrictive than their *GPL* counter-part. The *LGPL* specifically aimed at ameliorating licensing conflicts that arose when linking code to a non-(L)GPL library. Instead, the various versions of the *GPL* license require the system to change its license to the same version of the *GPL*, or else the component would not legally be able to be redistributed together with the project source code. Thus, it suggests a bias toward using less restrictive licenses even among the mostly used copyleft licenses. By the subsequent year (2003), a clear movement towards using less restrictive licenses can be seen with the wider adoption of

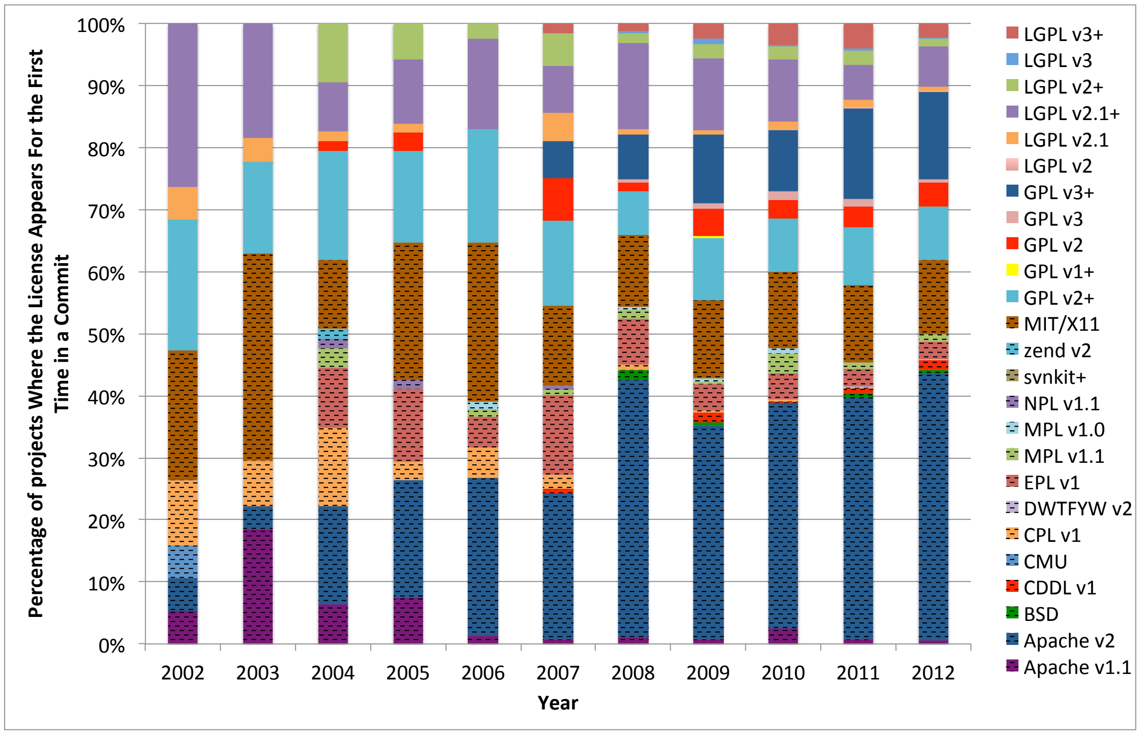


Figure 2.1: Relative license usage of the analyzed Java projects between 2002 and 2012 (dashed pattern representing permissive licenses).

the *MIT* license as well as the *Apache-1.1* license. Additionally, we observe that the *LGPL* is still prominent, while the *CMU*, *CPL-1.0*, and *GPL-2.0+* licenses were declining.

During the following five years (2004-2008), the *Apache-2.0*, *CDDL-1.0*, *EPL-1.0*, *GPL-3.0*, *LGPL-3.0*, and *DWTFYW-2* licenses were created. For the same observation period, Bavota *et al.* found that the Apache ecosystem grew exponentially in terms of number of lines of code of their source code [89]. This observation may contribute to the rapid diffusion of the *Apache-2.0* license among free/open source projects written in Java (other languages may not exhibit this behavior during the same time period). We observed a growth that resulted in the *Apache-2.0* license accounting for approximately 41% of licensing in 2008. Overall, we observe a tendency towards using permissive licenses, since ∼55-60% of licenses attributed were permissive during our time frame of analysis, excluding 2002.

Another interesting observation was that the newer version of the *GPL* (*GPL-3.0* or *GPL-3.0+*) had a lower relative usage compared to its earlier version until 2011. Additionally, the adoption rate was more gradual than for the *Apache-2.0* license that appears to supersede *Apache-1.1* license. However, the *LGPL-3.0* and *LGPL-3.0+* do not have more popularity than prior versions in terms of adoption, despite the relative decline of the *LGPL-2.1*'s usage starting in 2010. Our manual analysis of commits highlighted explicit reasons that pushed some developers to choose the *LGPL* license. For instance, a developer of the `hibernate-tools` project when committing the addition of the *LGPL-2.1+* license to her project wrote:

> *The LGPL guarantees that Hibernate and any modifications made to Hibernate will stay open source, protecting our and your work.*

This commit note indicates that *LGPL-2.1+* was chosen as the best option to balance the freedom for reuse and guarantee that the software will remain free.

Conversely, we observed the abandonment of old licenses and old license versions as newer free/open source licenses are introduced. For example, *Apache-1.1* and *CPL-1.0* become increasingly less prevalent or no longer used among the projects. In both cases, a newer license appears to replace the former license. While the *Apache-2.0* offers increased protections (*e.g.,* protections for patent litigation), the *EPL-1.0* and the *CPL-1.0* are the same license, with the only difference that IBM is replaced by the Eclipse Foundation as the steward of the license. Thus, the two licenses are intrinsically the same from a legal perspective, and most likely projects migrated from the *CPL* to the *EPL*; this would explain why the *EPL* adoption grew as the *CPL* usage shrunk.

Finally, we observed fluctuations in the the adoption of the *MIT* license. As the adoption of permissive licenses grew with the introduction of the *Apache-2.0* license, it first declined in adoption and was followed by growth to approximately its original adoption. Ultimately, we observed a stabilization of the *MIT* usage at approximately 10% starting in 2007.

Table 2.1: The results of the augmented Dickey-Fuller test to determine stationary or explosive trends in the license usage.

License	Stationary trend (p-value)	Explosive Trend (p-value)
Apache-1.1	0.14	0.86
Apache-2.0	0.98	**0.02**
BSD	0.73	0.27
CDDL v1	0.42	0.58
CMU	0.05	0.95
CPL-1.0	0.43	0.57
EPL-1.0	0.07	0.93
DWTFYW-2.0	0.99	**0.01**
MPL-1.0	0.90	0.10
MPL-1.1	0.32	0.68
NPL-1.1	0.55	0.45
svnkit+	0.78	0.22
zend-2.0	**0.01**	0.99
MIT	0.97	**0.03**
GPL-1.0+	0.05	0.95
GPL-2.0	0.67	0.33
GPL-2.0+	0.66	0.34
GPL-3.0	0.98	**0.02**
GPL-3.0+	0.69	0.31
LGPL-2.0	0.99	**0.01**
LGPL-2.0+	0.67	0.33
LGPL-2.1	0.35	0.65
LGPL-2.1+	0.54	0.46
LGPL-3.0	0.63	0.37
LGPL-3.0+	0.52	0.48

In order to determine whether the proportions for a given license exhibited a stationary trend, or a clearly increasing trend over the observed years, we performed ADF-tests. Results are reported in Table 2.1, where significant p-values (shown in bold face) in the second column indicate that the series is *stationary* (H_{0s} rejected), while significant p-values in the third column indicates that the series has an *explosive*, *i.e.,* clearly increasing, trend (H_{0e} rejected). The results indicate that:

- Almost no license is exhibiting a stationary trend. The results only show significant differences for the *zend-2.0* license, which is not particularly popular, and a marginal significance for *CMU*, *CPL-1.0* and *GPL-1.0+*.

- Confirming the discussion above, we have a clearly increasing trend not only for permissive licenses such as *Apache-2.0* and MIT but also for new versions of restrictive licenses facilitating the integration with other licenses (in particular, *GPL-3.0*, which eases the compatibility with the *Apache* license, as well as *LGPL-2.0*, which facilitates compatibility when code is integrated as a library). We also see an increase for *DWTFYW-2.0*, but this can be likely due to cases in which developers do not have a clear idea about the license to be used.

Summary for RQ$_1$. For the analyzed Java projects, we observed that generally using permissive licenses, like *Apache-2.0* and *MIT*, were more prevalent than restrictive licenses . Additionally, the permissiveness or restrictiveness of a license can impact the adoption of newer license versions, where permissive licenses are more rapidly adopted. Conversely, restrictive licenses seem to maintain a greater ability to survive in usage as compared to the permissive licenses, which become superseded. Restrictive (*GPL-3.0*) or semi-restrictive (*LGPL-2.0*) licenses that facilitate integration with other licenses also exhibit an increasing trend. Finally, we observed a stabilization in the license adoption proportions of particular licenses, despite the exponential growth of the GitHub code base.

2.1.2 RQ$_2$: What are the most common licensing change patterns?

We analyzed commits, where a license change occurred in one of the files, with a two-fold goal (i) analyze license change patterns to understand both the prevalence and types of licensing changes affecting software systems, and (ii) understand the rationale behind these changes. Overall, we found 204 different *atomic license change* patterns. To analyze them, we identified the patterns having the highest proportion across projects (*i.e.,* global patterns) and within a project (*i.e.,* local patterns). We sought to distinguish between dominant global patterns (Table 2.2) and dominant local patterns (Table 2.3) to study, on one hand, the overall trend of licensing changes and, on the other hand, to understand specific phenomena occurring in certain projects.

The global patterns were extracted by identifying and counting the presence of a pattern only once per project and then aggregating the counts over all projects. For instance, 823 projects in our dataset experienced at least one change (each) from *No license* → *Apache-2.0*, thus the final count (globally) for the pattern is 823. The most dominant global patterns were either a change from either no license or an unknown license to a particular license, or a change from either a particular license to no license or an unknown license. Table 2.2 shows the top ten global patterns. We observe that the inclusion of *Apache-2.0* was the most common pattern for unlicensed or unknown code. Clearly, this is likely due to the specific programming language (*i.e.,* Java) exploited by the sample of projects we quantitatively analyzed.

Table 2.2: Top ten global atomic license change patterns.

Top Patterns (Overall)	Frequency
no license or unknown → *Apache-2.0*	823
Apache-2.0 → no license or unknown	504
no license or unknown → *GPL-3.0+*	269
GPL-3.0+ → no license or unknown	181
no license or unknown → *MIT*	163
no license or unknown → *GPL-2.0+*	113
GPL-2.0+ → no license or unknown	111
MIT → no license or unknown	98
no license or unknown → *EPL-1.0*	94
no license or unknown → *LGPL-2.1+*	91
Top Migration Patterns Between Licenses	**Frequency**
GPL-3.0+ → *Apache-2.0*	25
GPL-2.0+ → *GPL-3.0+*	25
Apache-2.0 → *GPL-3.0+*	24
GPL-2.0+ → *LGPL-2.1+*	22
GPL-3.0+ → *GPL-2.0+*	21
LGPL-2.1+ → *Apache-2.0*	16
GPL-2.0+ → *Apache-2.0*	15
Apache-2.0 → *GPL-2.0+*	13
MPL-1.1 → *MIT*	11
MIT → *Apache-2.0*	11

Table 2.2 also shows the most common global migrations when focusing the attention on changes happened between different licenses. We observe that the migration towards the more permissive *Apache-2.0* was a dominant change among the top ten *atomic license changes* for global license migrations. An interesting observation is the license upgrade and downgrade between *GPL-2.0+* and *GPL-3.0+*. *GPL-3.0* is considered by the Free

Table 2.3: Top ten local atomic license change patterns between different licenses.

Pattern	Frequency
GPL-2.0+ → *GPL-3.0+*	36
GPL-2.0+ → *LGPL-3.0+*	15
LGPL-3.0+;*Apache-2.0* → *Apache-2.0*	12
GPL-3.0+;*Apache-2.0* → *Apache-2.0*	12
GPL-2.0+ → *LGPL-2.1+*	10
GPL-1.0+ → *LGPL-2.0+*	9
GPL-2.0+ → *GPL-3.0+*	9
GPL-3.0+ → *Apache-2.0*	8
GPL-3.0+ → *GPL-2.0+*	8
GPL-3.0+ → *LGPL-3.0+*	8

Software Foundation as a compatible license with the *Apache-2.0* license[1]. Due to the large usage of the Apache license in Java projects, this pattern is quite expected. However, the migration *GPL-3.0+* → *GPL-2.0+* is interesting, since it not only still allows for the project to be redistributed as *GPL-3.0* but also allows for the usage as *GPL-2.0*, which is less restrictive, as well.

Regarding the local patterns (Table 2.3), the frequencies were computed by first identifying the most frequent (*i.e.,* dominant) pattern in each project, and then counting the number of times a specific pattern is the most frequent across the whole dataset. For instance, the *GPL-1.0+* → *GPL-3.0+* pattern is the most frequent in 36 projects from our dataset. Table 2.3 summarizes the most common local migrations. The migrations appear to be toward a less restrictive license or license version. The low frequency of the *atomic license change* local patterns indicates that migrating licenses is non-trivial. It can also introduce problems with respect to reuse. For example, we observed a single project where *GPL-1.0+* code was changed to *LGPL-2.0+* a total of nine times. *LGPL* is less restrictive than *GPL*, when the code is used as a library. Thus, if parts of the system are *GPL*, the developer must comply with the more restrictive and possibly incompatible constraints.

Until now, we considered *atomic license changes* among any file in the repository. This was needed since most of the analyzed projects lack of a specific file (*e.g.,* license.txt)

[1]`http://gplv3.fsf.org/wiki/index.php/Compatible_licenses`

declaring the project license. To extract the declared project license, we considered a file in the top level directory named: *license*, *copying*, *copyright*, or *readme.* When just focusing on projects including such files, we extracted 24 different change patterns. Table 2.4 illustrates the top eight licensing changes between particular licenses (*i.e.,* we excluded no license or unknown license from this table) for declared project licenses. We only considered the top eight, since there was a tie between five other patterns or the next group of change patterns. We observe that the change from *Apache-2.0* $\rightarrow$ *MIT* was the most prevalent license change pattern, and the co-license of *MIT* with *Apache-2.0* is the second most prevalent one. Interestingly, this pattern was not dominant in our file-level analysis, although the Grounded Theory analysis provided us support for this pattern. The *MIT* license was used to allow commercial reuse, while still maintaining the open source nature of the project. It is important to note that the *Apache-2.0* license does facilitate commercial reuse; however, others clauses, like the one related to patents, may account for the preference towards the *MIT* license for commercial reuse.

The pattern of *GPL-2.0+* $\rightarrow$ *GPL-3.0+* (Top-3 in Table 2.4) was expected since it was tied for the most prevalent among global *atomic license changes.* Similarly, the patterns of *MIT/X* $\rightarrow$ *Apache-2.0*, *GPL-3.0+* $\rightarrow$ *Apache-2.0*, and *Apache-2.0* $\rightarrow$ *GPL-3.0* were also among the top eight global changes. Another notable observation is that license changes are frequently happening toward permissive licenses. Excluding the five changes from *Apache-2.0* $\rightarrow$ *GPL-3.0+* and *GPL-2.0+* $\rightarrow$ *GPL-3.0+*, the remaining changes for the top eight are either a licensing change from a restrictive (or copyleft) license to a permissive license or a licensing change between two different permissive licenses.

Summary for $\mathbf{RQ_2}$. The key insight from the analysis of *atomic license change* patterns observed on the studied Java projects is that the licenses tend to migrate toward less restrictive licenses.

Table 2.4: Top eight license change patterns in a declared license file of a project (license,copying,copyright, or readme file), excluding no license or unknown license.

Pattern	Frequency
Apache-2.0 $\rightarrow$ *MIT*	12
Apache-2.0 $\rightarrow$ *MIT*;*Apache-2.0*	8
GPL-2.0+ $\rightarrow$ *GPL-3.0+*	7
MIT $\rightarrow$ *Apache-2.0*	6
GPL-3.0+ $\rightarrow$ *Apache-2.0*	6
MIT;*Apache-2.0* $\rightarrow$ *Apache-2.0*	5
Apache-2.0 $\rightarrow$ *GPL-3.0+*	5
GPL-3.0+ $\rightarrow$ *MIT*	3

2.1.3 RQ$_3$: To what extent are licensing changes documented in commit messages or issue tracker discussions?

Table 2.5 reports the results of the identification of traceability links between licensing changes and commit messages/issue tracker discussions. We found a clear lack of traceability between license changes in both the commit message history and the issue tracker. In both data sources, we first extracted the instances (*i.e.,* commit messages and issue tracker discussions) where the keyword "license" appears **or** where a license name was mentioned (*e.g.,* "Apache"). In the former case, we are identifying potential commits or issues that are related to licensing, while the latter attempts to capture those related to specific types of licenses.

By using the first approach, we retrieved 70,746 commits and 68 issues; while looking for license names, we identified 519 commits and 712 issues. However, these numbers are inflated by false positives (*e.g.,* "Apache" can relate to the license or it can relate to one of the Apache Software Foundation's libraries). For this reason, we then looked for commit messages and issue discussions containing both the word "license" as well as the name of a license. This resulted in a drop of the linked commit messages to 399 and in zero issue discussions. Such results highlight that license changes are rarely documented by developers in commit messages and issues.

We also investigated whether relevant commits and issues could be linked together. We linked commit messages to issues when the former explicitly mentions fixing a particular issue (*e.g.,* “Fixed #7” would denote issue 7 was fixed). We observed that this technique resulted in a large number of pairs between issues and commits; thus, our observation of a lack of license traceability is not simply an artifact of poor traceability for these projects. To further investigate the linking, we extracted the commit hashes where a license change occurred and attempted to find these hashes in the issue tracker’s comments. Since the issue tracker comments contain the abbreviated hash, we truncated the hashes appropriately prior to linking. Our results indicated only one match for an open issue and zero matches for closed issues.

Finally, we attempted to link changes to issues by matching date ranges of the issues to the commit date of the license change. The issue had to be open prior to the change and if the issue had been closed the closing date must have been after the change. However, we did not find any matches with a date-based approach.

Summary for RQ$_3$. For the analyzed Java projects, both the issue tracker discussions and commit messages yielded very minimal traceability to license changes, suggesting that the analysis of licensing requires fine-grained approaches analyzing the source code.

2.1.4 RQ$_4$: What rationale do these sources contain for the licensing changes?

In this section, we firstly present the taxonomy that resulted from the open coding of commit messages and issue tracker discussions. This analysis has been performed on 1,637 commit messages and 486 issue tracker discussions from 1,160 projects written in seven programming languages, and aims at modeling the rationale of license adoption and changes. Secondly, we present our findings when looking at the commits that introduce *atomic license changes* in the analyzed Java projects.

Analyzing Commit Messages and Issue Discussions Table 2.6 reports the categories obtained in the open coding process. In total, we grouped commit messages and issue tracker discussions into 28 categories, and organized them into seven groups that will be described in detail in the rest of this section. Additionally, 430 commits and 161 issue

Table 2.5: Traceability between licensing changes and commit messages or Issue tracker discussion comments.

Data Source	Linking Query	Links
Commit Messages	Commits with the keyword “license”	70746
	Commits containing new license name	519
	Commits containing new license name and the keyword “license”	399
Issue Tracker Comment Matching	Comments from closed issues containing the keyword “license”	0
	Comments from closed issues containing the new license	0
	Comments from closed issues containing the new license and the keyword “license”	0
	Comments from open issues containing the keyword ”license”	68
	Comments from open issues containing the new license	712
	Comments from open issues containing the new license and the keyword “license”	16
Issue Tracker Date-based Matching	Closed comments opened before license change and closed before or at license change	197
	Open comments open before the license change	2241
	Comments from closed issues open before the license change and closed before or at the license change with keyword “license”	0
	Comments from open issues open before the license change with keyword “license”	0
Issue and Commit Matching	Comments in closed issues containing the keyword ”Fixed #[issue_num]”	66025
	Comments in open issues containing the keyword ”Fixed #[issue_num]”	3407
	Comments in closed issues containing the commit hash where the license change occurs	0
	Comments in open issues containing the commit hash where the license change occurs	1

Table 2.6: Categories defined through open coding for the Issue tracker discussion comments and Commit notes.

Category	C		C++		C#		Java		Javascript		Python		Ruby		**Overall**	
	I	C	I	C	I	C	I	C	I	C	I	C	I	C	**I**	**C**
Generic License Additions																
Choosing License	1	0	0	0	0	0	6	0	2	0	1	0	1	0	**11**	**0**
License Added	1	22	3	19	0	15	25	75	22	34	9	34	1	33	**59**	**232**
License Change																
License Change	2	14	1	8	1	5	3	14	4	9	2	6	2	18	**15**	**74**
License Upgrade	0	4	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**4**
License Rollback	0	0	0	0	0	0	0	1	0	0	0	2	0	0	**0**	**3**
Removed Licensing	0	3	0	3	0	4	0	6	1	8	0	2	0	3	**1**	**29**
Changes To Copyright																
Copyright Added	0	6	0	3	0	2	0	0	0	2	0	2	0	0	**0**	**15**
Copyright Update	2	24	0	7	1	6	5	89	2	7	2	4	1	8	**13**	**138**
License Fixes																
Link Broken	7	0	2	0	0	0	1	0	16	0	1	0	19	0	**46**	**0**
License Mismatch	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**
Fix Licensing	4	2	0	1	0	2	1	3	2	0	0	1	2	1	**9**	**10**
License File Modification	0	11	0	8	0	14	0	0	1	11	1	7	1	29	**3**	**80**
Missing Licensing	1	1	0	0	0	3	2	0	7	0	12	0	4	1	**26**	**5**
License Compliance																
Compliance Discussion	1	9	0	5	1	1	0	1	0	3	0	1	0	0	**2**	**20**
Derivative Work Inconsistency	0	0	0	0	0	0	0	0	0	0	1	0	0	0	**1**	**0**
Add Compatible Library	0	1	0	0	0	0	3	0	0	0	0	2	0	0	**3**	**3**
Removed Third-Party Code	3	13	1	8	0	1	0	1	0	2	0	4	0	3	**4**	**32**
License Compatibility	0	0	0	0	0	0	6	0	0	0	0	0	0	0	**6**	**0**
Reuse	1	1	1	0	0	0	17	0	1	0	1	0	0	0	**21**	**1**
Dep. License Added	0	0	0	0	0	0	0	0	0	0	0	1	0	0	**0**	**1**
Dep. License Issue	2	0	0	0	0	0	1	0	1	0	0	0	0	0	**4**	**0**
Clarifications/Discussions																
License Clarification	2	0	2	1	1	0	19	0	2	1	4	0	2	0	**32**	**2**
Terms Clarification	0	0	0	0	0	0	5	0	2	0	0	0	0	0	**7**	**0**
Verify Licensing	0	0	0	0	0	0	0	1	0	0	0	1	0	0	**0**	**2**
License Agreement	0	0	0	0	0	0	2	0	2	0	0	0	0	0	**4**	**0**
Request to License Project																
Licensing Request	1	0	0	0	0	0	0	0	4	0	0	0	6	0	**11**	**0**
License Output For The End User																
Output Licensing	0	0	1	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**

discussions identified by means of pattern matching as potentially related to licensing were classified as false positives. This is mainly due to the wide range of matching keywords that we used for our filtering to identify as many commits/issues as possible. Finally, for 16 commits and two issue discussions that were related to licensing, it was not possible, based on the available information, to perform a clear categorization. Thus, they were excluded from this study.

In the following, we discuss examples related to the various groups of categories.

Generic license additions. This group of categories concerns cases in which a license was added in a file, project or component where it was not present, as well as discussions related to choosing the license to be added in a project. One typical example of commit message, related to the very first introduction of a software license into the repository, mentioned:

> *"Added a license page to TARDIS."* [42]

Other commit messages falling in this category were even precise in reporting the exact license committed into the repository, *e.g.,*:

> *"Add MIT license. Rename README to include rst file extension."* [35]

Finally, commit messages automatically generated by the GitHub's licensing feature were present, *e.g.,*:

> *"Created LICENSE.md."*

While commit messages show the addition of a license to a project, they do not provide the rationale behind the specific choice. This can be found, sometimes, in the discussions carried out by the developers in the issue trackers to establish the license under which their project would be released. For example, one of the issue discussions we analyzed was titled "Add LICENSE file" [43] in the project *web-workshops*, and the issue opener explained the need for (i) deciding the license to adopt and (ii) involve all projects' contributors in such a decision:

"A license needs to be chosen for this repo. All contributors need to agree with the chosen license. A list of contributors is enclosed below."

Doubts and indecision about which license to adopt were also evident in several of the issue discussions that we manually analyzed:

"What license to use? BSD, GNU GPL, or APACHE?" [9]

Interestingly, one developer submitted an issue for the project *InTeX* entitled "Dual license under LGPL and EPL" [18] that related to adding a new license to balance code reuse of the system, while avoiding "contagious" licensing (the term "contagious" was used by the original developer of the system). The developer commented:

"Your package is licensed under GPL. I'm not a lawyer but as far as I understand the intention of the GPL, all LaTeX documents compiled with the InTeX package will have to be made available under GPL, too. [...] I think, you want users to publish changes they did at your code. A dual license under LGPL and EPL would ensure that a) changes on your code have to be published along with a binary publication and b) that your code can be used in GPL and non-GPL projects. See JGraphT's relicensing for more background."

This response demonstrates a potential lack of understanding regarding the license implications to compiled LaTeX and proposes dual-licensing as a solution. However, the original developer also indicates a lack of legal background and is not willing to offer a dual-license based on his understanding stating:

"Thank you for your interest. I not a lawyer myself either, but my intentions are:
1. I want changes to the source code of InTeX to be made available so that others can benefit from them too.
2. I do not want any "contagious" copyright of documents compiled with InTeX. However, I've always thought of InTeX as a (pre)compiler, and given

> *this GPL FAQ answer, I think licensing the compiler's source code under GPL does not limit or affect the copyright of the documents it is used to process. Unless you can prove me wrong about this, I will close this issue."*

Thus, the developer responds by providing his understanding of the *GPL* by referencing a response by GNU regarding compiled Emacs[2]. However, the developer does indicate an openness to adding a new license if the *GPL* would in fact be applied to generated LaTeX documents. This example is particularly interesting, since it shows the original developer's rationale for picking the *GPL* as well as the difficulty that developers have with respect to licensing.

License Change. This group of categories concerns cases in which (i) a licensing statement was changed from one license towards a different one; (ii) a license was upgraded towards a new version, *e.g.,* from *GPL-2.0* to *GPL-3.0*; (iii) cases of license rollback (*i.e.,* when a license was erroneously changed, and then a rollback to the previous license was needed to ensure legal compliance); and (iv) cases in which for various reasons developers removed a previously added license.

Most commit messages briefly document the performed change, *e.g., "Switched to a BSD-style license"*, *"Switch to GPL"*. Some others, partially report the rationale behind the change:

> *"The NetBSD Foundation has granted permission to remove clause 3 and 4 from their software"*

The commit message explains that permission has been granted for the license change by the NetBSD Foundation. However, the committer does not explain the reason for the removal of the two clauses. Other commits are instead very detailed in providing full picture of what happened in terms of licensing:

[2]`http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#CanIUseGPLToolsForNF`

"Relicensed CZMQ to MPLv2 - fixed all source file headers - removed COPYING/COPYING.LESSER with GPLv3 and LPGv3 + exceptions - added LICENSE with MPLv2 text - removed ztree class which cannot be relicensed - (that should be reintroduced as foreign code wrapped in CZMQ code)."

The commit message from the project *CZMQ* [8] is very informative, reporting the former license (*i.e., GPL-3.0* and *LGPL-3.0*), the new license (*i.e., MPL-2.0*), and the removal of the ztree class since it is unable to be relicensed. This license change demonstrates a move towards a more permissive license, which has been shown to be prevalent in our study of Java projects.

We also found commit messages reporting the rationale behind specific license changes, such as the following commit from the project *nimble* [25]:

"Change in project License from AGPL 3.0 to Apache 2.0 prior to first public release. Several factors influenced this decision the largest being community building and making things as easy as possible for folks to get started with the project. We don't however believe Open Source == Free and will continue to investigate the best way to commercialize this. Restrictive copy-left licenses aren't however the answer."

While the developers want to enable external developers to reuse the system, they are also interested in commercializing the software product. The developers acknowledge that copy-left licenses do not meet their needs.

For *License Rollback*, we observed that the project *PostGIS* reverted back licensing to a custom license [30]. The commit does not offer rationale since it simply states:

"Restore original license terms."

From the analysis of the commit emerged that the author had re-licensed the system under the *GPL* earlier and subsequently reverted back the licensing to his own custom license. However, it is not clear if this rollback was due to a misappropriation of *GPL*, an incompatibility in the system, or to other factors.

Additionally, we found commit messages illustrating that license removals do not necessarily indicate that the licensing of the system was removed. For instance:

"Removing license as it is declared elsewhere" [34]

"Remove extra LICENSE files
One repository, one license. No need to put these on the box either." [10]

"Remove licenses for unused libraries" [7]

In these cases, the system contains redundant or superfluous license files that can be removed. This observation highlights that strictly analyzing the license changes that have happened in the history of a software system could (wrongly) suggest that the system has migrated toward closed-source. The third commit message, instead, indicates that licenses were removed due to unused code, which required those licenses. Such cases, in which a project is adopting unnecessary licenses due to third-party libraries no longer needed, should be carefully managed since it may discourage other developers to reuse the project, especially if the unnecessary licenses are restrictive.

Changes to Copyright. This group of categories includes commits/issues related to simple changes/additions applied to the copyright statement, like copyright year, or authors. Changes to a list of author names occur to indicate names of people who provided a substantial contribution to the project, therefore claiming their ownership. Previous work indicated that often such additions occur in correspondence of large changes performed by contributors whose names are not mentioned yet in the copyright statement [129]. Changes to copyright years have also been previously investigated, and are often added to allow claiming right on source code modified in a given year [100].

License Fixes. This group of categories is related to changes in the license mainly due to various kinds of mistakes or formatting issues, as well as to cases in which a licensing statement was accidentally missing (note that this is different to cases of license addition in which the license was originally intended to be absent from the project).

For example, in this group, we observed cases of issues discussing *license mismatch*, where developers found conflicting headers or conflicts between the declared license and the license headers. In the former case, a developer posted an issue to the project *gtk-sourcecompletion*'s issue tracker [15]:

> *"The license states that this is all LGPL-3, but the copyright headers of the source files say otherwise (and some are missing). Is this intentional, or should these all be under the same license? I've included licensecheck output below."*

Subsequently, the issue poster listed files in the system with *GPL*, *LGPL*, and no copyright. Additionally, he indicated cases where the Free Software Foundation address was incorrect as well. We observed a similar situation in another project: a developer opened the issue "LICENSE file doesn't match license in header of svgeezy.js" [40] to svgeezy's issue tracker and stated:

> *"The LICENSE file specifies the MIT license, but the header in svgeezy.js says it's released under the WTFPL. Which is the correct license?"*

In this second case, we observe that the declared license and source header are not consistent. However, the issue has not been resolved at the time of writing this paper and so we cannot report the resolution or any feedback offered by the original developers of the system.

Other interesting cases are the ones related to the fix of *missing licenses.* Often developers are made aware of missing licenses via the issue tracker by projects' users reporting the issue. Sometimes, the complete project may be unlicensed, leading to discussions like the one titled "GNU LGPL license is missing" from the project *rcswitch-pi* [33]:

> *"Under which license is this source code published? This project is heavily based on wiring-pi and rc-switch: rc-switch: GNU Lesser GPL wiring-pi: GNU*

Lesser GPL The GNU Lesser GPL should be added:

http://www.gnu.org/licenses/lgpl.html"

Based on the project's characteristics (*i.e.,* its foundations on previously existing projects), the developer recommends the addition of the missing LGPL license.

The commits and issues falling in the *License File Modification* category are related to changes applied to the license file type or name. For example, developers may change the license file from the default `LICENSE.md` file generated by GitHub to a *.txt* or *.rtf.* Additionally, developers change the file name often to make it more meaningful as illustrated in this commit message of the project *Haml* [17]:

> *"Renamed the LICENSE to MIT-LICENSE so you don't have to open the file to find out what license the software is released under. Also wrapped it to 80 characters because I'm a picky [edited]"* (quote edited for language)

Other typical changes concern the renaming of the `COPYRIGHT` file to `LICENSE` or the move of the license file in the project's root directory. These cases do not indicate changes towards a different license or in general any change to the license semantics, but only in the way in which the license is presented.

License Compliance. This group of categories is probably the most interesting to analyze, and concerns categories related to discussions and changes because of license compliance. Specifically, other than generic compliance discussions, there are cases in which (i) a derivative work's legal inconsistency was spotted or discussed; (ii) a compatible library is added to replace another incompatible library from a licensing point of view; (iii) third-party code is completely removed when no legally-compliant alternative was possible; (iv) cases of discussion related to license compatibility in the context of reuse; and (v) cases in which an added dependency or an existing dependency has conflicts with the current license.

A very interesting example is the issue discussion entitled *"Using OpenSSL violates GPL licence"* in the project *SteamPP* [38]. Surprisingly, the developer of the project initially commented:

> *"gnutls and libnss have terrible documentation and I don't consider this a priority issue anyway. If you would like to submit a pull request, then be my guest."*

Despite this initial reaction, the OpenSSL library was replaced by Crypto++ within a week in order to meet the licensing requirements.

Examples of third-party libraries removed due to licensing issues are also prevalent in commit messages, *e.g.,*:

> *"Remove elle(1) editor, due to an incompatible license."* [22]

The incompatibility in this case was due to *elle*'s clause explicitly reporting: *"NOT be sold or made part of licensed products."*. Additionally, we saw the commit from the project *wkhtmltopdf-qt-batch*, where files were removed due to a recommendation by the project's legal staff: *"Remove some files as instructed by Legal department"* [45]. This shows that license compliance may not be always straightforward to developers and that they may need to rely on legal council in order to determine whether licensing terms have been met.

We also observed changes in the system's licensing aimed at satisfying compliance with third-party code in the project *gubg* [16]:

> *"Changed the license to LGPL to be able to use the msgpack implementation in GET nv."*

Similarly, we found issue tracker discussions about conflicting licenses or about the compatibility of licenses between the project and third-party libraries. Interestingly, there was an issue opened by a non-contributor of the project *android-sensorium* [4], stating:

> *"Google Play Services (GMS) is proprietary, hence not compatible with GNU LGPL. (The jar inside the Android library referred to in the project.properties).*

F-Droid.org publishes the ....o3gm package, but we cant publish this without removing this library."

Thus, the license incompatibility not only created a potential license violation for the project but also prevented the non-contributor from cataloging the system among projects hosted on F-Droid [11], a well-known forge of free/open source Android apps.

Additionally, we observed issues related to *reuse*, where one contributor suggests a dual license to allow for greater reuse in other applications. The contributor of the project *python-hpilo* [32] stated,

"Due to incompatibility between GPLv3 and Apache 2.0 it is hard to use python-hpilo from, for instance, OpenStack. It would therefore be helpful if the project code could also be released under a more permissive license, like for instance Apache 2.0 (which is how OpenStack is licensed)"

The other contributors subsequently utilized the thread to vote and ultimately agreed upon the dual license. Not only does this example indicate the consideration for reuse but it also demonstrates that licensing decisions are determined by most of the developers and not by a single developer. It is also important to note that *GPL-3.0* and *Apache-2.0* are not consider incompatible by the Free Software Foundation.

Conversely, we also observed an interesting discussion in which the issue posted in the project *patchelf* [28] asked *"Is it possible for you to change GPL to LGPL? It would help me using your software."*. The developer posting the question was developing a system licensed under the *BSD* license with which *GPL* would not be compatible. A contributor refused to change licensing by stating: "*GPL* would not be compatible". Moreover, one of the contributors explained that changing licensing is non-trivial by responding:

"It wouldn't be easy to change the license, given that it contains code from several contributors, who would all need to approve of the change."

Again, this response highlights the importance for all contributors to approve a license change. However, reaching an agreement among all contributors might be far from trivial, due to personal biases developers could have with respect to licensing [153].

We also observed a case related to derivative work, where the license differed from the original system's licensing (category: *Derivative Work Inconsistency*). A developer created the issue *"Origin and License Issue"* for the project *tablib* [41] to which he offered support, but first noted:

> *"While tablib is MIT-licensed, there are several potential provenance and license issues with Oo, XLS and XLSX formats that tablib embeds. I have collected some of these potential issues here. This is at best ... byzantine. [...] https://bitbucket.org/ericgazoni/openpyxl/ is reported as being derived from PHPExcel which is LGPL-licensed at https://github.com/PHPOffice/PHPExcel but openpyxl is not LGPL but MIT-licensed. If this is really derived then there is a possible issue as the license may be that of the original not of the derivative."*

The issue poster lists the various components used with their licensing to point out incompatibility issues, and in particular those related to the derivative code that the system utilizes.

Clarifications/Discussions. This group of categories contains issues related to clarifying the project's licensing, the terms or implications of the licensing, and the agreement between contributors made in a Contributor License Agreement (CLA). *License Clarification* were about the actual license of the project and typically occurred when the system did not contain a license file (*i.e.,* a declared project license). For example, one project's user created the issue *"Please add a LICENSE file"* for the Mozilla's project *123done* [2] stating:

> *"The repo is public, but it's not easy to find out how I'm allowed to use or*

> *share the code.*
>
> *Could you add a LICENSE file to make it easier for users to understand how you'd like it to be used?"*

Similarly, another project, *pyelection*, has the issue *"What license is this code released under?"* [31] with no further comments from the poster. Thus, we observe that developers use the issue tracker as a mean to understand the licensing and request an explicit licensing file.

Another surprising issue discussion is related to understanding the terms of a license. The issue was posted to the *neunode*'s issue tracker [23] by an external developer looking to reuse the code and asked:

> *"We are impressed with what you've done with neu.Node and are interested in using it for offline mapping applications. However, we work at a company that has more than 1M$ in revenue. Your license terms say MIT for companies with less than 1M$ in revenue (which is not an approach I've seen before). Please could you clarify the license terms for a company that is larger that that? We're trying to make some decisions on our direction at the moment, so a quick response would be appreciated if possible."*

Interestingly, the license terms set conditions based on the money value of the company looking to reuse the code. In this case, the external developer's company exceeds the threshold. The original developer indicates that his software is intended to benefit the developer community as a whole, and more specifically students and individuals. The original developer gave two options: (i) a large check without maintenance support, or (ii) detail descriptions of the product, a compelling argument for giving a free license to reuse the system, and acknowledgment in the description. Thus, the original developer is not interested to financial gain (though, he could reasonably be convinced at the right price), but rather wants to support the open source community and receive credit for his work. It is important to note that this license is not a free/open source license.

We identified a category of *License Agreement*. This scenario arises when an external developer to the project submits some code contribution to the project, and the project contributors require that developer to complete a Contributor License Agreement (CLA) to avoid licensing/copyright disputes. We observed a discussion related to updating the textual information of the project's CLA with respect to country designations [6]. Similarly, in our previous Java study [152], a developer submitted a patch, but it could not be merged into the system until that developer filled out the CLA [19]. A CLA makes it explicit that the author of a contribution is granting the recipient project the right to reuse and further distribute such contribution [91]. Thus, it prevents the contributed code from becoming a ground for a potential lawsuit.

Request to License Project. This group contains issue discussions in which a developer asks for a license or a license file. While these are similar to *reuse*, it differs since the developers do not necessarily state that they want to reuse the system, since it is possible that they want to contribute as well. Thus, these are more generic requests for the developer to attribute a license to the system without explaining the reason for such a request. For example, we found the issue titled "No license included in repository" for the project *jquery-browserify* [20] in which the poster commented:

> *"Would you consider adding a license to the repository? It's currently missing one and according to TOS.*
>
> [*Not posting a license*] *means that you retain all rights to your source code and that nobody else may reproduce, distribute, or create derivative works from your work. This might not be what you intend.*
>
> *Even if this is what you intend, if you publish your source code in a public*

> *repository on GitHub, you have accepted the Terms of Service which do allow other GitHub users some rights. Specifically, you allow others to view and fork your repository.*
>
> *If you want to share your work with others, we strongly encourage you to include an open source license.*
>
> *If you don't intend on putting a license up that's fine, but if you do want to use an open source license please do so. I'd be happy to fork/PR for you if you just let me know which license you want to put in (MIT/BSD/Apache/etc.)"*

This comment demonstrates that licensing also impacts derivative work and can prevent other developers from contributing to a system. This is an important distinction, since findings and prior work [152, 153, 137] demonstrate that licensing could be an impediment to reuse and not an impediment to contribute towards a project/system.

License output for the end user. This category describes a unique case where an issue was posted regarding the output of the license to the end user. The issue stated:

> *"This output could be read by monitoring tools, for example to automatically warn about expiration (although Phusion also emails expiration warnings, the desired upfront time for the warning is not configurable like that)."* [27]

Unlike the previous categories, this issue relates to end user licensing the software. The contributor of the system suggests the inclusion of a feature to aid in monitoring the license expiration. Interestingly, this category shows that developers also consider licensing from the impact on the "client" using the system. This aspect of understanding the impact of licensing on the "client" or end user has also been unexplored in prior studies.

2.1.4.1 Analysing Commits Implementing Atomic License Changes in Java Systems

In this analysis, we specifically targeted commit messages where a licensing change occurred so that we could understand the rationale behind the change. We did not apply a keyword for these commit messages since we knew they were commits related to changes in licensing. When reading these commits, we also included the *atomic license change* pattern that was observed at that particular commit to add context. We observed new support for the existing categories and the results are reported in Table 2.7. We refer to new support as commit messages indicating new rationale for the existing categories.

As for the *License Change* group of categories, we observed general messages indicating a license change occurred and in some cases explicitly stating the new license, such as the following commit messages:

> *"Rewrite to get LGPL code."*
>
> *"Changed license to Apache v2"*

These two commit messages do not offer rationale, but they at least indicate the new license that has been attributed to the system. So, a developer inspecting the change history would be able to accurately understand the particular license change.

Since we observed many instances of *no license* $\rightarrow$ *some license*, the prevalence of *License Added* was expected. However, these *License Added* commit messages resembled the *License Change* messages since they often did not include a clear rationale (i.e., while being part of the *License Added* category, their *level of detail* was similar to the *License Change* category). For example, a developer asserted the *Apache-2.0* license to the headers of the source files across his project, but his commit message simply stated:

> *"Enforce license"*

Table 2.7: Categories defined through open coding for the commit messages in which a license change occurred. The results are presented as part of full the taxonomy.

Category	Commits
GENERIC LICENSE ADDITIONS	
Choosing License	0
License Added	63
LICENSE CHANGE	
License Change	9
License Upgrade	1
License Rollback	1
License Removal	19
CHANGES TO COPYRIGHT	
Copyright Added	0
Copyright Update	1
LICENSE FIXES	
Link Broken	0
License Mismatch	0
Fix Missing Licensing	9
License File Modification	0
Missing Licensing	1
LICENSE COMPLIANCE	
Compliance Discussion	0
Derivative Work Inconsistency	0
Add Compatible Library	0
Removed Third-Party Code	1
License Compatibility	0
Reuse	0
Dep. License Added	0
Dep. License Issue	0
CLARIFICATIONS/DISCUSSIONS	
License Clarification	0
Terms Clarification	0
Verify Licensing	0
License Agreement	0
REQUEST TO LICENSE PROJECT	
Licensing Request	0
LICENSE OUTPUT FOR THE END USER	
Output Licensing	0

In the case of *License Removal*, we observed that licenses were removed due to code clean up, files deletion, and dependencies removal. For example, we observed the removal of the *GPL-2.0* license with the following commit message,

> *"No more smoketestclientlib"*

It indicates the removal of a previously exploited library. Additionally, licenses were removed as developers cleaned up the project.

Fix Missing Licensing is related to a license addition, but it occurred when the author intended to license the file, but forgot either in the initial commit or in the commit introducing the licensing. For example, one commit message stated:

> *"Added missing Apache License header."*

This indicates that the available source code may inaccurately seem unlicensed.

Additionally, *License Upgrade* refers to license change, where the version of the license is modified to the most recent. In this particular case, we observed a change from *GPL-2.0+* to *GPL-3.0+*. The commit message stated:

> *"...Change copyright header to refer to version + 3 of the GNU General Public License and to point readers at the + COPYING3 file and the FSF's license web page."*

While the commit message describes the version change, it does not supply rationale. Instead, the message is a log of the changes.

An important observation from the second round of our analysis was the ambiguity of commit messages. For example, we observed a commit classified as *Copyright Update* stating,

> *"Updated copyright info."*

However, this commit corresponded to a change in licensing from *GPL-2.0* to *LGPL-2.1+*. This case both illustrates the lack of detail offered by developers in commit messages, and it illustrates that an update can be more significant than adding a header or changing a copyright year.

Since we sampled commits from all Java projects, it was infeasible to sample a larger representative number of commit messages. Thus, augmenting the second round by considering commits in which an *atomic license change* occurred benefited the taxonomy by targeting relevant commits better. However, we were able to sample statistically representative sample sizes in this work due to pre-filtering the projects. The results corroborate the representativeness, since we observed the same categories.

Another important observation that appears to support the supposition from our traceability analysis that developers remove licensing related issues from the issue tracker is that we found links that were removed in the period of time between our crawling and our data analysis. These were categorized as *Link Broken* and amounted to 45 of the overall issues. It is also possible that these cases represent developers that utilize external bug tracking systems as well.

Summary for RQ$_4$. While our open coding analysis, based on grounded theory, indicated some lack of documentation (*e.g.,* prevalence of false positives) and poor quality in documentation with respect to licensing in both issue tracker discussion and commits notes, we formally categorized the available rationale. We also found that the rationale may be incomplete or ambiguously describe the underlying change (*e.g.,* "Updated copyright info" representing a change between different licenses). Finally, we observed that issue trackers also served as conduits for project authors and external developers to discuss licensing.

2.2 Practitioner Perspective on Initial Licensing and Changing Licensing

At some point, the creators of free/open source software must choose a license that: 1) expresses the developers' philosophy; 2) meets their deployment goals, and 3) is consistent with the licenses of the components reused by that software. However, choosing a license is not an easy process. Developers do not necessarily have a clear idea on the exact consequences of licensing (or not licensing) their code under a specific license; for instance, developers ask questions on Question & Answer (Q&A) websites looking for advice on how to redistribute code licensed with a dual license among the other issues (e.g., question 2758409 in Stack Overflow [110] and question 139663 in the StackExchange site for programmers [140]). Also, the problem of license incompatibility between components is not trivial (see [105] for a detailed description of this problem).

During the evolution of a software system, its license might change. In Chapter 2.1, we empirically showed—for software hosted in GitHub— license changes occur in free/open source software. Stemming from the results that we previously captured by analyzing licensing and their changes in software repositories [152], *the goal of this work is to understand when and why changes in licensing happen.* Specifically, this work reports the results of a survey of 138 developers with the aim of understanding (i) *when* developers consider adding a license to their project, (ii) *why* they choose a specific license for their projects, and (iii) *factors* influencing license changes. The 138 participants are the respondents from a set of 2,398 invitees, i.e., 5.75% of the invitees. We identified such developers by sampling 16,221 Java projects on GitHub, and then subsetting to 1,833 projects where the license changed over time. Of these 138 developers, 52 developers offered insights to the aforementioned questions, while the remaining developers reinforced that licensing decisions are not necessarily made by all contributors, but by a subset that are the copyright holders.

The main findings of this study are as the following:

1. Developers frequently license their code early, but the main rationale for delaying licensing is usually to wait until the first release;

2. Developers have strong intrinsic beliefs that affect their choice of licenses. Also, free/open source foundations, such as the Apache Software Foundation, the Free

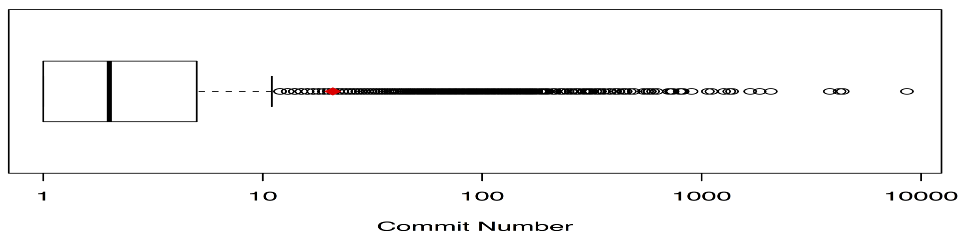


Figure 2.2: Distribution of commits with license adoptions (log scale).

Software Foundation, and the Eclipse Software Foundation exert a powerful influence on the choice of a license;

3. We observed that the change of a license(s) of a system is predominantly influenced by the need to facilitate reuse (mostly in commercial systems);

4. Developers experience difficulties in understanding the licensing terms and dealing with incompatible licenses.

Complete details of the design of the study can be found in our ICSME'15 publication [153]

2.2.1 RQ$_1$: When and why do developers first assert a licensing to their project?

Fig. 2.2 shows the distribution of the number of commits in which licenses were introduced into the projects within our dataset (e.g., a license introduced during the tenth commit will be represented by the number 10). We present the raw commits in log scale due to outliers from large commit histories. At least 25% (first quartile) of the projects were licensed in the first commit (Fig. 2.2). The median was also at two commits and third quartile was at five commits. This observation indicates that free/open source projects are licensed very early in the change history with over 75% of the projects having a license by the fifth commit. Assuming (but this might not be always the case) that the observed history corresponds to the entire project history, this result suggests that licensing is important to developers. It is interesting to note that the mean commit number for adding a license is 21 and the maximum value is 8623 commits. These two values are indicators of a long tail with a small number of projects that consider licensing late in the change history.

Table 2.8 reports the responses to Question 3 (Q3) of our survey in which we tried to ascertain the rationale behind the initial project licensing. 30.8% of developers indicated that the community influences the initial licensing. One explanation for the high prevalence of this response is that certain free/open source communities stipulate and enforce that a particular license must be used. For example, the Apache Software Foundation requires that its projects or the code contributed to their projects are licensed under the *Apache-2.0* license. Instead, the *Free Software Foundation* promotes the use of the *GPL* and *LGPL* family of licenses.

19.2% of developers chose the license with the goal of making their project reusable in commercial applications. These responses also indicate a bias toward more permissive licenses that facilitate such usage while restrictive licenses can discourage such usage, since they require that a system is licensed under the same terms. This finding provides a partial explanation for the tendency toward using more permissive licenses that we observed in our previous work [152].

The results of our survey also show that licensing-related decisions are impacted by inherent developer bias. 15.4% of developers supplied answers that we categorized as moral-ethical-beliefs. An example of this category was the response by one developer indicating, "*I always use GPL-3.0 for philosophical reasons.*" Similarly, a different developer echoed this comment stating "*I always licence GPL, moral reasons.*"

Satisfying a dependency constraint (i.e., the need to use a license based on the license of dependencies) was a relevant reason (9.6% - 7.7% picking the explicit option and 1.9% with an "Other" response categorized as dependency constraint). This result is important, since little work has been done to analyze licensing across software dependencies. This problem also poses challenges in both identifying all of the necessary dependencies as well as the license(s) of those dependencies. Some automated build frameworks like *Maven* [87] or *Gradle* [14] attempt to ameliorate this difficulty by listing dependencies in a file that drives the building process (e.g., Project Object Model file in Maven). However, licensing is not a required field in those files.

The remaining answers to this question described situations in which the license was inherited by the initial founders and persisted over time. Also, the companies have policies to specifically dictate a licensing convention. In the latter case, the respondent indicated that "*company (...) policy is Apache-2.0*" (company name omitted for privacy). It was also interesting to see that nobody choose a license based on requests by outsiders.

Lastly, we identified a category in licensing changes that related to license adoption and not changes. 7.7% of developers respond to our question on licensing changes that indicated the license was missing and it was added in a later commit. For this case, we added (License Addition) to the category for Q4 in Table 2.8. The developers noted that "*Setting the license was just forgotten in the first place*" and "*Accidentally didn't include explicit licence in initial commit*". These cases are also important, since it can create inconsistencies within the system or mislead non-contributors that the project is unlicensed or licensed under incompatible terms. This result further reinforces that developers view early license adoption as important, but the lack of a license may be a mistake.

Summary for RQ$_1$ We observed that developers consider licensing early in the change histories of free/open source projects. While there are projects that assert a license after a larger number of commits, 75% of our dataset had a license asserted within the first five commits. Thus, the data suggests that most of the projects adopt licenses among the very first commit activities. The initial licensing is predominantly influenced by the community to which a developer is contributing. Subsequently, commercial reuse is a common factor, which may reinforce the prevalence of permissive license usage. While reuse is a consideration, non-contributors do not seem to impact the initial licensing choice. We also found that the inclusion of a particular dependency can impact the initial licensing of a project.

2.2.2 RQ$_2$: When and why do developers change the licensing of their project?

Fig. 2.3 shows the distribution of when licenses were changed in the projects within our dataset (i.e., *Some license→Some Other License*). As in the previous section, we present the raw commit number in which the changes occurred (log scale due to outliers from large commit histories). Interestingly, the minimum value was the second commit (i.e., a license changed right after its addition in the first commit). More generally, 25% of license changes occur in the first 100 commits. The median value is 559 commits while the mean is 3,993 commits. The third quartile (2,086 commits), quite smaller than the mean, suggests a long tail of license changes occurring late in the projects' change histories. The maximum commit number with a license change was commit 56,746. Numbers at this extreme would cause the larger mean value compared to the median. Overall, the data suggests that certain projects change licenses early in the change history; however, the license changes are much more prevalent in later commits.

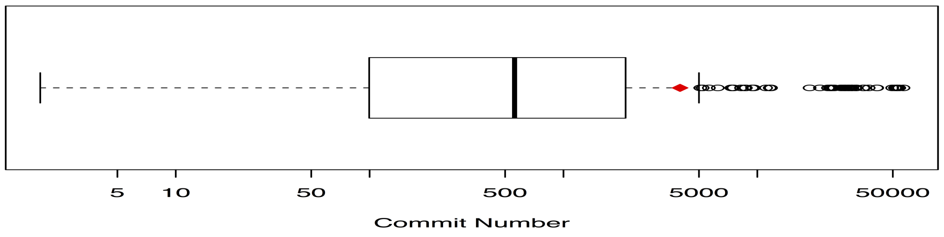


Figure 2.3: Distribution of commits with license changes (log scale).

Table 2.8 shows the responses to Question 4 (Q4) of our survey in which we investigated the rationale behind license changes. Allowing reuse in commercial software was the most common reason behind licensing changes (32.7%). This option was also the second most prevalent for choosing the initial license (19.2% of developers). Combining these two results, it is clear that the current license of a project is heavily affected by its need to be reused commercially. As previously stated, this result qualitatively supports the observation from our previous work [152], where we observed that projects tend to migrate toward less-restrictive licenses.

7.7% of developers changed licensing due to community influence. This response was a more significant factor for the initial choice in licensing, but it further emphasizes the impact that a community can assert. One developer commented, "*community influence (contributing to Apache's projects)*". Similarly, two developers commented about the influence the Eclipse Foundation exercised over license changes in their projects. Interestingly, one developer reported: "*I wanted to use the most common one for OSS Java projects*". This response suggests that a particular license may pick up more momentum and spread for a particular language. Interestingly, we observed that 7.7% of the developers were willing to change the licensing due to requests from non-contributors. The fact that this response was more prevalent for changing licensing than choosing the initial license may be influenced by outsiders waiting until a project is stable or mature before inquiring about particular licensing.

We also observed that both the change in license(s) of a dependency or using a new dependency prompted developers to change licenses (5.8% of developers for both cases). This observation further demonstrates the difficulty or impact that dependency can have with respect to licensing. It also suggests that there could be inconsistencies between the licensing of a system and its dependencies.

Moral-ethical-beliefs are also a reason for 5.8% of developers. Interestingly, we observed both the beliefs of developers and beliefs of a philantropist, who is funding the project's development. While one developer acknowledged, "*I simply wanted to pick a 'free' license and chose Apache without much consideration,*" another developer indicated that "*Philanthropic funders encouraged us to move to GPL3, as well as our own internal reflection on this as we came to understand GPL3 better.*" In the former example, it is notable that the developer's concern was not the impact of the *Apache* license in particular, but primary motivator was any *free* license (i.e., free/open source license). The latter indicates that the individuals funding the projects can influence the licensing. While the developers were not coerced to change to the *GPL-3.0*, they were still influenced by the beliefs of the individuals funding the system's change history.

Summary for RQ$_2$ We observed that developers change licensing later in the change history of the free/open source projects. While there are projects that change licensing early, our first quartile was 100 commits and third quartile was 2,086 commits, demonstrating more substantial development occurred before changing licensing. Developers seem to change licensing to support reuse in commercial systems. While community influence still impacts changing licensing, it appears to be a less significant factor with respect to license adoption. Based on our survey results, the reasons behind changing licensing are more diverse and more evenly distributed among the topics than we observed in the selection of the initial license.

2.2.3 RQ$_3$: What are the problems that developers face with licensing and what support do they expect from a forge?

Table 2.9 shows the results for Questions 5-7 (Q5-Q7) that investigate both the problems that developers experience with licensing and expected licensing support from the forge.

In Q5, we investigated the problems related to licensing that developers have experienced. 23 out of 52 developers (44.2%), explicitly mentioned "No problem" in the "Other" field. For those who recognized problems, the main reason was the inability of others to use the project due to its license (17.3%). Since developers consider this a problem, it suggests that developers are interested in allowing broad access to their work. However, they may be constrained due to desired protections (e.g., patent protection from *Apache-2.0* or *GPL-3.0*) or external factors, like the licensing of dependencies (external since the developers cannot change those licenses).

Additionally, developers indicated that choosing the correct license was difficult for them (13.5%). The litigious nature of the text of these licenses can lead to misinterpretations by developers. For example, the Apache Foundation states on their webpage that "*The Apache Software Foundation is still trying to determine if this version of the*

Apache License is compatible with the GPL" [5]. Additionally, 5.8% developers indicated that they experienced misunderstandings with respect to license compatibility. To make matters worse, 9.6% of the developers experienced compatibility problems with dependencies. Therefore, developers not only faced difficulty while determining the appropriate license, but they also misunderstood the compatibility among licenses and experienced incompatibility between their project's licensing and a desired dependency's licensing.

Developers also experienced difficulties with their users misinterpreting or not understanding the terms of their license. One developer stated that "*Users do not read/understand the license, even though it is a most simple one.*" This result poses two possible problems — either users (i.e., developers looking to reuse the code) ignore the actual licensing text or they struggle to interpret even the easier licenses. The former would demonstrate a bigger problem in that users do not take licensing seriously, while the latter demonstrates that the difficulty in understanding licensing is more extensive than just very litigious licenses. Reinforcing the second scenario, another developer noted the problem was "*Just the usual challenges of talking with potential commercial partners who do not understand the GPL at all*". By phrasing the comment with *the usual challenges*, it suggests that the developer had repeated experience with partners unable to understand licensing. This is not necessarily an isolated case, but rather potentially widespread experience shared by other developers.

Regarding the support provided by the forge, in this case GitHub, we investigated the impact of a feature added to help document the license of a project—see Q6 in Table 2.9. This feature was added as a response to the criticism from some practitioners [130]. While 36.5% of developers did not have access to the feature at the time they created their project, the interesting result is that more than half (51.9%) of developers were not influenced by the availability of such a tool. Additionally, the "Other" responses indicated that the feature would not have had an impact on their choice (3.8%) and a single developer specifically chose not to license her project, leading to a combined 58% of developers that were unaffected by this feature. Thus, our data suggests that this GitHub feature did not

Table 2.8: Developer Involvement and Rationale for Choosing or Changing Licenses. Survey Results for Questions 1 to 4.

Question/Answer	**#D**	**%**
Q1. Were you involved in changes occurring to parts of the system that underwent license changes?	**138**	
Yes	75	54.3%
No	63	45.7%
Q2. Were you involved in determining the license or change in license of the project or some of its files?	**138**	
Yes	76	53.7%
No	62	46.3%
Q3. How did you determine/pick the initial license for your project or files in your project?	**52**	
Dependency constraint	4	7.7%
Community influence (e.g., contributing to Apache projects)	16	30.8%
Requests by non-contributors to reuse your code	0	0%
Interest of reuse for commercial purposes	10	19.2%
Other (please specify)	22	42.3%
— Closed-source	1	1.9%
— Company-policy	2	3.8%
— Dependency-constraint	1	1.9%
— Inherit-license	3	5.8%
— Moral-ethical-belief	8	15.4%
— Project-Specific	2	3.8%
— Social-trend	2	3.8%
— None	3	5.8%
Q4. What motivated or caused the change in license?	**52**	
License of dependencies changed	3	5.8%
Using a new library imposing specific licensing constraints	3	5.8%
Allow reuse in commercial software	17	32.7%
Requests by non-contributors to reuse your code	4	7.7%
Other (please specify)	25	48.1%
— Change-to-license-text	2	3.8%
— Community-influence	4	7.7%
— Fix-incorrect-licenses	1	1.9%
— Improve-clarity	1	1.9%
— Missing-license (License Adoption)	4	7.7%
— Moral-Ethical-belief	3	5.8%
— More-permissive-license	1	1.9%
— New-license-version	2	3.8%
— Personal-Preference/Project-specific	1	1.9%
— Private-to-public-project	1	1.9%
— Promote-Reuse	1	1.9%
— Unclear	1	1.9%
— None	3	5.8%

Table 2.9: Problems with Licensing and Expected Support from a Forge. Survey Results for Questions 5 to 7

Question/Answer	**#D**	**%**
Q5. What problems (if any) have you experienced due to license selection in terms of code reuse?	**52**	
My license was not compatible with desired dependencies	5	9.6%
Others were unable to use my project unless I re-licensed it	9	17.3%
A dependency changed licenses ad was no longer compatible	1	1.9%
There was a misunderstanding of compatibility between licensing terms of two licenses	3	5.8%
Choosing the correct license was difficult/confusing	7	13.5%
Other (please specify)	27	52.9%
— Code-unavailability	1	1.9%
— Lack-of-undersanding-by-Users	2	3.8%
— Unique-New-License	1	1.9%
— No problems	23	44.2%
Q6. Did GitHub's mechanism for licensing impact your decision on licensing your project?	**52**	
Yes, it caused me to license my project	3	5.8%
No, I already planned on licensing	27	51.9%
No, I did not want to license at project creation	1	1.9%
Such a mechanism was not yet available when I created my project	19	36.5%
Other (please specify)	2	3.8%
— No impact	2	3.8%
Q7. What kind of support would you expect from the forge/GitHub to help you managing licenses and licensing compatibility issues in your software?	**11**	
None	10	90.9%
License Checker and License Selection Wizard	1	9.1%

affect/influence developers when licensing (or not) software hosted in GitHub.

Finally, we received 11 responses to our optional question (Q7) concerning whether forges should provide features that assist the licensing of their software. Since GitHub has been criticized by practitioners [130] for a lack of licensing consideration, this question seeks to understand the features that practitioners expect from a forge to this end. 10 out of 11 participants answered "*None*". Of those 10 developers, only one explained that a third party tool should handle license compatibility analysis. The respondent indicated that the ideal tool would utilize the various forges and build frameworks to be a dependency graph of license compatibility stating the following:

> *"This is the job of a 3rd party tool IMO since neither github nor forge do or should own all open source deps. A 3rd party tool ideally would know about github, bitbucket, etc + poms and pom license fields, etc and form a comprehensive dep-graph license compat view given a node."*

Another developer noted, "*None. From our perspective it really isn't that hard to put copyright and licence notices in our source files.*" This comment is interesting since it conflicts with results from Q4, where developers indicated that licenses were sometimes missing or an incorrect license was used.

The only developer wishing support from the forge indicated a desire for a license compatibility checker and a license selection wizard. This developer commented the desire for two particular features stating the following:

> *"1) License compatibility checker - verify the license of your project with the license of included support software (gems, libraries, includes) and alert user to potential conflicts. This could also be used for the use case that you want to adopt a piece of software to add to an existing project - is it compatible? 2) License selection wizard - when you begin a project, the wizard can ask you a series of questions (do you want to allow commercial use, do you require mods to be licensed the same as original, etc) and then suggest a license for the project."*

While only one developer wanted support from the forge, this single developer's comments seem to address many of the problems and difficulty with respect to licensing for which we found evidence in Q6 of the survey.

Summary for RQ$_3$ Although 44.2% of the developers surveyed indicated that they have not experienced problems with licensing, the remaining respondents provided a diverse set of answers. They primarily were related to license incompatibility or difficulty understanding the licensing. Lastly, the survey indicated that GitHub's mechanism to encourage or aid in licensing was not necessary or unavailable to the surveyed developers. We also found that most developers did not expect support from the forge, but one did indicate the desire for a third-party tool. However, one developer did express interest in forge's support, and the comments aligned with our results regarding problems that developers actually faced.

2.3 Discussion

Lack of Traceability of Licensing The analysis of the commit messages and issue tracker discussions highlighted that the information offered with respect to licensing choice/change is very often quite limited. A developer interested in reusing code would be forced to check the source code of the component to understand the exact licensing or to ask for clarification (using the issue tracker, for example). Additionally, the reason behind the change is not usually well documented. This detail is particularly important when a system uses external/third-party libraries, since a license may change during the addition or removal of those libraries.

The lack of traceability of licensing changes is also important for researchers investigating software licensing on GitHub. While we cannot generalize to other features, it does suggest that commit message analysis may be largely incomplete with respect to details of the licensing-related changes made during that commit. One way to achieve this for developers is to take advantage of summarization tools such as *ARENA* [126] and *ChangeScribe* [96, 116]. While *ARENA* analyzes and documents licensing changes at release level, *ChangeScribe* automatically generates commit messages; however, using *ChangeScribe* would require extending it to analyze licensing changes at commit level. Another option is that forges (and software tools in general) verify that every file contains a license and that every project properly documents its license (this feature could be optional).

Lack of Clear or Standard Way to Document the License Similarly, we observed that external developers would request a license, since the projects appeared to be unlicensed; however, a subset of these requests were due to licensing being attributed in a different manner than external developers expected (*e.g.,* part of the `gemspec` file for Ruby projects and not a `LICENSE` file). We also observed developers adding license files to parent directories as opposed to headers in the source code as well as appending the license name to the license file (*e.g.,* `LICENSE` would be renamed `LICENSE.MIT`). This way of declaring a license is particularly used in GitHub project where the system asks the developer(s) to choose a license, when a project is created, and then it creates the `LICENSE` file in the project's root directory.

These observations indicate a lack of standardization in how licensing is expressed among both projects in the same language and projects across different languages. It suggests that **developers need a standardized mechanism to declare the license of a software project. Third-party tools or forges could support developers by maintaining this standardized documentation automatically.**

Intrinsic Beliefs of the Developers. The first important observation is that the participants have a bias toward free/open source licensing from an ethical perspective. 52% of the respondents indicated (Q6) that they planned on licensing the project prior to creation; only 6% of the respondents (Q6) were influenced to license their project due to GitHub's licensing feature (i.e., a combo list with license names). Similarly, the "Other" responses regarding the reason for a project's initial licensing (Q3) indicated a sense of obligation. For example, one developer said: "*It was the only moral and ethical choice*".

Delayed Licensing. Developers do not necessarily have to decide to open source from the beginning and delay doing it. While we empirically observed early license adoption in general, one developer wrote in an email that they waited to choose a license: "*this project just didn't have a license on day 1 and it was added at first release.*" Similarly, one developer responded to the survey that licensing changed due to "*change private to public project*". This observation suggests that licensing is still important to these developers, but it may not be considered relevant until the project reaches a certain level of maturity. Thus, there is the need for tools to add and verify licensing information of a system at any given point in time.

Community and Organizational Influence. Our results indicate that communities, and in particular free/open source foundations (such as the Apache Software Foundation, Eclipse Foundation, and the Free Software Foundation) exert powerful influence on the choice of a license by its developers. About 31% of the participants responded that initial licensing is done by following community's specific licensing guidelines. Improving or developing on top of existing software from a foundation mostly requires using the same license aligning with foundation's philosophy.

Difficulty Understanding Licenses. The survey stresses the need for aid in explaining licenses and the implications of their use. About 20% of the respondents highlighted that licensing is confusing and/or hard to understand (Q5): 13.5% of respondents indicated that developers—both the authors and the users—find licensing confusing or difficult (Q5), and 6% of developers also noted that there were misunderstandings between license compatibility. Additionally, one "Other" respondent stated, "*Users do not read/understand the license, even though it is a most simple one*," which suggests that developers experienced misunderstanding whether on their own or by users.

Reuse for Commercial Distribution. The results regarding licensing changes indicated that commercial usage of code is a concern in the open source community. We found that practitioners used permissive licenses to facilitate commercial distributions, in some cases they change to a more permissive license for this purpose.

Dependency Influence. A software system must choose its dependencies to avoid conflicts due to incompatibilities between the system's license(s) and the depending components' license(s). Similarly, others will choose to use a particular software system based on its license. Thus, the change of a license in a system has the potential of creating a chain reaction: those that use it might need to change their license, or drop it as a dependency; for the system changing license, the potential pool of reusable components will change accordingly—it might need to drop a different dependency, or it might be able to add a dependency with a previously incompatible license.

Forge's Support. Most of our respondents do not expect any licensing support from the forge. It is likely that the individuals that benefit the most from licensing support in the forge are those who are looking to reuse software. This is supported by our results that indicate that the license(s) of dependencies is an important consideration, since it might impact the ability to reuse the dependency or require a change of the license(s) of the software that uses it. Thus, compliance-oriented features may aid developers to ensure they can legally reuse software.

Finally, our results demonstrate that external factors like community, license prevalence, and licenses of dependencies have an important impact on licensing.

A feature provided by the forge to support domain suggested licensing could benefit practitioners. Since developers indicated that licensing is difficult, a more informative feature could help practitioners determine the appropriate licensing. For instance, the current licensing support feature provided by GitHub feature is not particularly informative for developers. Basically, it provides a link to `choosealicense.com`, but does not provide further guidance to the developer. Also, it does not cover issues related to compatibilities at all. Moreover, applications within the same domain may be utilizing some of the same dependencies or require similar grants for redistribution and reuse. To better support developers, a forge could include a domain analysis feature to detect similar applications [120] and suggest to the developer/maintainer the license used by similar systems (if no other criteria has been considered, such as community or dependencies).

2.4 Bibliographical Notes

The empirical studies referenced in this Chapter were performed in collaboration with other members of the SEMERU group at William and Mary and researchers from the Universidad de los Andes, University of Sannio, University of Lugano, and the University of Victoria:

- **Vendome, C.**, Bavota, G., Di Penta, M., Linares-Vásquez, M., Germán, D., and Poshyvanyk, D., "License Usage and Changes: A Large-Scale Study on GitHub", Empirical Software Engineering (EMSE), June 2017, Volume 22, Issue 3, pp. 1537–1577.
- **Vendome, C.**, Linares-Vásquez, M., Bavota, G., Di Penta, M., Germán, D., and Poshyvanyk, D., "When and Why Developers Adopt and Change Software Licenses“, in Proceedings of 31st IEEE International Conference on Software Maintenance and Evolution (ICSME'15), Bremen, Germany, September 29 – October 1, 2015, pages 31–40.
- **Vendome, C.**, Linares-Vásquez, M., Bavota, G., Di Penta, M., Germán, D., and Poshyvanyk, D., "License Usage and Changes: A Large-Scale Study of Java Projects on GitHub“, in Proceedings of 23rd IEEE International Conference on Program Comprehension (ICPC'15), Florence, Italy, May 18-19, 2015, pages 31–40.

Chapter 3

Licensing Bugs

In this chapter, we present a new type of bug that we coined as *licensing bug*, which impact the ability to distribute software and reuse existing software. Prior empirical studies [85, 100, 103, 105, 135, 136, 137, 149, 152, 153] showed that (i) developers face difficulties in choosing the appropriate license for their work as well as in understanding the legal implications behind licenses, and (ii) incompatibilities between licenses can represent a serious threat from a legal perspective for both free/open source and commercial software. These two pieces of evidence combined suggest that licensing issues are likely to occur in software projects and that their effect could be non-negligible. However, there is limited evidence in the literature about these licensing issues that developers generally face and of their legal/technical implications [100]. We refer to such issues as *licensing bugs*, mostly related to legal incompatibilities of licensed software and to the breach of guidelines that can prevent software from being distributed or modified (*e.g.,* by preventing a patch from being accepted). While licensing bugs may not be as pervasive as other types of bugs, they have the ability to block development and prevent software from being released, and in the worst case, may result in monetary implications.

In order to understand these *licensing bugs*, we preformed a large-scale qualitative study aimed at characterizing *licensing bugs*, with the goal of understanding the types of licensing bugs developers face, their legal and technical implications, and how such bugs are fixed. In this chapter, we aim at answering the following research question:

- *What are the types of licensing bugs faced by developers?*

Specifically, we investigate the types of licensing bugs that developers discuss (and try to solve) in various sources of communications. We aim to define a catalog of licensing bugs, which includes the impacted stakeholders, the implications and ability to address the licensing bugs. To this aim, we mined 59,426 discussions (from issue trackers and mailing lists) carried out by software developers and likely related to licensing bugs. Then, we manually analyzed — via an open-coding procedure — a statistically significant sample of 1,200 discussions, randomly selected from the initial set of 59,426 discussions. During the manual analysis, we transcribed the type of licensing bugs, their implications, and possible fixes.

As a result of the open coding, we present a catalog of seven categories and 21 sub-categories of licensing bugs. During our analysis, we observed licensing bugs that had not been previously addressed in the research community. These licensing bugs relate to jurisdiction of laws, non-source/non-binary artifacts, and ecosystem policies. We also observe that the lack of freeness can prevent the distribution of software. In the context of this paper, *freeness* refers to a community's expectations for a license not to impose unreasonable restrictions on the distribution and the modification of software. In the global scope of free/open source software, *freeness* refers to the utilization of a free/open source license; however, a community can define more stringent guidelines (*i.e.,* not all free/open source licenses may be viewed as free by a particular community). The term *freeness* is not equivalent to *gratis* (*i.e.,* being without charge or free in a monetary sense), but it refers to the ability for others to reuse, redistribute, and modify some entity (typically but not limited to source code and binaries). Additionally, we observed that the stakeholders related to licensing bugs vary from developers to patch integrators and lawyers.

3.1 Design of Study

We aim at addressing the following question: *What are the types of licensing bugs faced by developers?* We investigate licensing bugs in FOSS projects, by characterizing them in terms of types, difficulties they pose, their implications on legal and technical aspects, and the ability to (or the extent to which) licensing bugs can be addressed (or resolved). To this, we mined developers' discussions from (i) issue trackers hosted on GitHub [12] and Bugzilla [55], and (ii) mailing lists specialized in legal aspects of software products. All the data used in this study are available in the attached appendix [81].

We defined a catalog of licensing bugs, highlighting their type, the stakeholders that are impacted by the bugs, the implications of the licensing bugs, and the extent to which they can be addressed.

3.1.1 Identification of Candidate Licensing Bugs from Developers' Discussions

The issue trackers and mailing lists provided us with complementary insights with respect to licensing bugs. The issue trackers facilitate reporting of the bug, while the mailing lists provide deeper discussion related to licensing bugs with respect to both legality and community guidelines. We performed an open coding of the data source to build a catalog of licensing bugs.

We focused on mailing lists dealing with the discussion of legal aspects of software products, and in particular: Apache's *legal-discuss* [47], Debian's *debian-legal* [57], Fedora's *fedora-legal-list* [71], Gnome's *legal-list* [72], and OpenStack's *legal-discuss* [79]. All the considered mailing lists have publicly available archives storing all messages exchanged through them. Once the archives were downloaded, we automatically identified discussions linking each message to the corresponding email thread by exploiting the "Message-id" field in the "In-reply-to" field. This allowed us to analyze each message in the context of the discussion, thus favoring an easier interpretation of the legal issue being discussed. In total, we had 45,019 candidate messages with licensing bugs.

Concerning the issue trackers, we mined the 136 Bugzilla issue trackers listed in the Bugzilla's installation list [51]. To identify issues relevant to licensing, we queried the Bugzilla's search functionality using the keyword "*license*". This led to the selection of 2,125 candidate licensing bugs. Additionally, we mined the issue trackers of 86,032 projects hosted on GitHub. We selected projects that (i) are not a fork (to avoid duplications), and (ii) have at least one star, watcher, or fork (to perform a first cleaning of "toy" projects) [113]; we then locally cloned these repositories and filtered out all projects with less than fifty commits, again with the goal of removing trivial projects. Out of the selected 258,057 projects, we considered 86,032 projects that had an issue tracker. In particular, we crawled the candidate licensing-related issues (including the issue title, description, metadata, and related discussion) from each issue tracker by using a keyword search mechanism exploiting specific licensing keywords (*e.g., copyright*) or license names (*e.g., GPL*) [152]. This keywords-based search resulted in 12,282 candidate licensing-related issues.

After gathering the data, we randomly selected issues and messages, and performed a pre-coding. During this step, all authors coded 100 randomly selected issues to create an initial set of codings, and to ensure that they were utilizing a consistent criteria when coding issues and messages. Since we observed a high percentage (approximately 67%) of false positives in this initial pre-coding (*i.e.,* discussions unrelated to licenses), we performed a pre-filtering to remove false positives. To this aim, one author read a randomly selected issue or message to determine whether it contained a licensing bug. If the author concluded that it was a false positive, the issue or discussion was discarded from the set of issues/messages to be coded. This process was performed repeatedly for each data source until reaching 400 documents containing a candidate licensing bug from each source, for a total of 1,200 issues/messages. The number of 400 documents per data source represents a statistically significant sample with 95% confidence level and a confidence interval of ±4.42% for Bugzilla issues, ±4.82% for GitHub issues, and ±4.88% for mailing list messages (we ensured having a confidence interval smaller or equal to ±5%).

3.1.2 Definition of the Catalog of Licensing Bugs

To devise our catalog of bugs, we manually inspected the 1,200 selected mailing list and issue tracker discussions (from now on, generally referred to as "discussions") via an open-coding procedure [124]. The goal was to categorize the type of licensing bugs. We divided the 1,200 documents among the authors such that each document was coded by at least two-authors for two-author agreement, obtaining a list of 21 initial categories (each discussion belonging to a single category). While we had attempted to filter data to remove false positives, there were still 78 documents identified as false positives (6.5%) and 21 tagged as unclear (1.83%). These two categories are omitted from the presented catalog in the results.

Each discussion was randomly assigned two of the authors to independently categorize it. After each round, the two authors discussed and resolved conflicts, and generated categories from the codings. Before solving conflicts, the coding achieved an agreement ratio of 65%. To determine whether such agreement level was due to chance, we computed the Cohen's kappa inter-rater agreement coefficient [94], which resulted to 0.53 (moderate agreement).

After, we defined a higher level of abstraction over the set of 21 initial distinct categories using a card-sorting approach [139]. To this aim, the 21 categories were presented in a spreadsheet, and two of the authors independently reordered the categories and clustered them. After that, they discussed the clusters they had created, and converged to a common solution. In the end, this resulted in grouping license bugs into seven distinct categories, listed in the 21 sub-categories previously identified (Table 3.1).

3.2 Catalog of Licensing Bugs

We present and discuss the catalog of licensing bugs. Specifically, we present a description of each licensing bug, the stakeholders *most directly impacted* by the particular licensing bug (i.e., the individuals most directly responsible for them and their remediation), the implications (both legal and technical) of the bugs, and the extent to which they can be addressed/fixed. Also, we provide examples of discussions related to that particular category of licensing bug. While previous studies on licensing [153] mostly focused on developers, we consider different stakeholders impacted by the bugs:

1. **Integrators:** developers that are reusing free/open source software within their own systems;

2. **Package Maintainers:** developers responsible for maintaining packages and integrating patches or bug-fixes into existing distributions of software;

3. **Distributors:** any individual or entity distributing software, which can be a single developer, a free/open source foundation/community (*e.g.,* Apache), or a company (*e.g.,* IBM);

4. **Developers:** this category relates to developers in general, without making specific distinctions;

5. **Lawyers:** interpreting licensing in terms of the existing laws or responsible for drafting new licenses;

6. **Lawmakers:** people responsible for writing and passing laws;

7. **Community:** either people involved in a specific open source community, or the open source community as a whole;

8. **Trademark Holders:** companies or individuals that have trademarked their name or logo (*e.g.,* as a means of brand protection).

The taxonomy is composed of 21 distinct sub-categories organized in 7 distinct high-level categories (Table 3.1). Due to space limitations, we only discuss a subset of the sub-categories (14). The complete taxonomy description and frequencies of each category can be found in the attached appendix [81]. In the reported examples, we have adjusted formatting (*e.g.,* removing mid-sentence newlines) in some cases, but the wording remains unchanged. It is important to remark that the results discuss the interpretation of developers and/or legal practitioners. Therefore, it is possible that the legality of these interpretations or discussions may change (*e.g.,* new interpretations can cause new legal precedents in the U.S.A.), or the enforceability may change in different jurisdictions. Our results have the purpose of discussing these interpretations and implications from the perspective of the developers and legal practitioners within particular open source communities.

3.2.0.1 Laws and Their Interpretations

This category comprises licensing bugs related to different interpretations of licenses provided by different people and organizations as well as to how licenses are interpreted under different jurisdictions.

3.2.0.2 What is Copyrightable?

Description: Developers experience issues when trying to understand the implications or scope of copyright coverage. We observed discussions regarding copyright and emulation, legal consequences of game console emulators, executed programs, or donated code. We also observed discussions related to textual updates to copyrights, *e.g.,* copyright year upgrade. We found that the scope of the coverage is not necessarily clear for developers. While software is copyrightable (the basis of free/open source software licensing), higher-level design and ideas *may* fall out of the scope of copyright law. Potential disagreements on the coverage of copyright protection can potentially lead to the infringement of another

Table 3.1: Catalog of Licensing Issues with (# of Discussions).

Laws and Their Interpretations	**(169)**
What is Copyrightable?	(20)
What is a Derivative Work?	(30)
What is the Jurisdiction?	(14)
License Interpretation	(44)
Clarification Issues	(61)
Policies of the Ecosystem	**(133)**
Community Guidelines	(34)
Freeness: can I reuse it?	(99)
Potential License Violations	**(129)**
Compatibility between Licenses	(86)
Other Types of License Violations	(43)
Non-source Code Licensing	**(45)**
Documentation Licensing	(16)
Font and Media Licensing	(29)
Licensing Content	**(485)**
Incorrect Licensing	(37)
License Inconsistencies	(183)
Licensing Missing	(207)
License Textual Issues	(46)
Outdated/Obsolete Licensing	(12)
Other Intellectual Property Issues	**(63)**
Rights to Use a Contribution	(35)
Patent-Related Issues	(20)
Trademark	(8)
License Semantics	**(76)**
Dual Licensing	(16)
License/Clause Implication	(60)

entity's copyright. In an example below, we demonstrate the difficulty to understand the coverage of copyright for game emulation.

Stakeholders: Developers and lawyers.

Implications: It has been demonstrated that copying as few as 27 lines of code can constitute copyright infringement [123]. Violating these laws can result in legal action against developers and prevent the distribution or reuse of the system violating the copyright (see the subsection on *License Violation*). However, the scope of copyright law (similar to patent and trademarks) can only be addressed by lawyers, and to truly define the scope it would require lawmakers to more explicitly or rigidly define the extent of copyright; alternatively, in precedent-based legal systems, prior court rulings may also serve to define the scope and implications of copyright.

Examples: In terms of emulation, one individual commented in the thread that emulation of the games themselves can be illegal,

> "That second case is pretty much where we stand with a *lot* of game console emulators out there – the only way to get data to use with them is to break the law. Wonderful." [68]

However, another developer posits whether it still complies with copyright law explaining:

> "Is it illegal if I own a game cartridge, and dump it? That part probably isn't; US copyright law, at least, give me permission to make a backup copy." [68]

Such an issue requires legal consultation since emulations could be legal in certain cases, but creating a game emulator could violate a company's IP. Thus, the determination of freeness for an emulator would depend upon the dependencies to *run* the emulator and the potential IP infringement.

3.2.0.3 What is a Derivative Work?

Description: A derivative work is partially owned by the copyright author on which it is originally based. In order to be able to distribute a derivative work, its creator needs a license from the work on which it is based. In fact, one of the most important features of free/open source licenses is that they should allow the creation and redistribution of derivative works. However, some licenses place important restrictions on such redistribution (*e.g.,* the GPL family of licenses requires that any derivative work must also be distributed under the GPL). Therefore, when a software system reuses another product, the question on whether the former is a derivative work of the latter is critical. This issue might affect not only software, but also non-software artifacts, like images.

Stakeholders: Distributors, package maintainers, integrators.

Implications: If a product A is a derivative work of product B, the license of product B can impose restrictions on how product A can be used and licensed to others. Such

restrictions could mean that product A might not be used or licensed as expected. Or, if product A is redistributed, it could result in legal liability due to the risk of copyright infringement. On the other hand, if product A is not a derivative work of product B, it might be possible to redistribute product B along product A, as long as the license of product B is properly satisfied. Thus, evaluation of whether a work is considered derivative work can either result in license violations or incompatibilities if the derivative work is considered non-free or subject to the terms of an incompatible restrictive license. To cope with such bugs, when possible developers can use/ask for license exceptions [151].
Examples: Some systems provide clarifications on whether something is considered a derivative work of another. For example, there has been a long discussion on what is a derivative work of the Linux kernel. Linus Torvalds clarifies this in the COPYING file,

> "NOTE! This copyright does not cover user programs that use kernel services by normal system calls - this is merely considered normal use of the kernel, and does not fall under the heading of "derived work". Also note that the GPL below is copyrighted by the Free Software Foundation, but the instance of code that it refers to (the Linux kernel) is copyrighted by me and others who actually wrote it." [77]

However, there is still plenty of disagreement on this issue [148].

In a discussion regarding firmware being derivative work, one individual cites US copyright law and how it defines derivative work and collections. One developer replies:

> "However – by this definition, the linux kernel is very definitely a derivative work, and the firmware is content which has been incorporated into the kernel." [63]

The developer that presented the legal text follows up saying:

> "The kernel (I assume as a whole) is a derivative work of what? I would argue that the kernel is a compilation of what executes on the host CPU with other parts (boot logos, fonts, and firmware data). The executable part may be a derivative in part of Adam Richter's code, but that does not necessarily make the kernel as a whole a derivative of his work." [64]

The difference in interpretations demonstrates the difficulty distinguishing the extent that derivative work applies as well as distinguishing between derivative work and collective work.

3.2.0.4 What is the Jurisdiction?

Description: We observed licensing bugs related to the differences in laws across countries. Copyright, trademark and patent laws are national in scope. While the World Intellectual Property Organization (WIPO) attempts to unify intellectual property law around the world, specific issues of the law might be different from one country to another. Also, software is affected by other laws, such as restrictions on trade. For example, moral rights are enshrined in the copyright laws of Europe and Canada, but they are not present in the copyright laws of the United States.
Stakeholders: Distributors, developers, and lawmakers.
Implications: These differences might not be large, but they might have an important impact on the ability to create and distribute software.They might also have an impact on any potential litigation (and the choice of jurisdiction). In fact, we observed that clauses related to choice of jurisdiction were a controversial topic within Debian in terms of their impact on software's freeness. However, the distribution may be impacted by external factors like trade restrictions to a particular country or distribution of what a country considers sensitive material. While organizations or communities may want to facilitate global reuse, the organizations and individuals must comply with these trade laws. Also, government funded work (*e.g.,* supported by a grant) may require release of the software for public domain domestically (*i.e.,* within the funding country), but the government may impose restrictions internationally, which can only be addressed by policy changes by the funding entity. Thus, these cases are predominantly external to developers, but impose restrictions that must be considered.
Examples: In *debian-legal*, there is question regarding a texture created by NASA and

the possibility to reuse the texture. While one person indicates that the copyright would depend on whether it was done by NASA itself or a contractor (*e.g.,* a contractor of the Jet Propulsion Laboratory), another person comments:

> “While a work may be in the public domain in the U.S., it may be under copyright elsewhere. So, *e.g.,* while works by the U.S. government may be public domain in the U.S., they may remain under copyright in other countries. However, the U.S. government may license their works and thus give permissions with respect to these foreign copyrights.“ [62]

Thus, the copyright issues become further complicated by the fact that government created works might only be released domestically into the public domain [46].

Another post in *debian-legal* suggests that Gentoo was being utilized as a base distribution in Cuba, since Debian is blocked due to the US embargo with Cuba. Another person responds:

> “It is my understanding that it is illegal under US law to knowingly export software (free software or otherwise) from the US to Cuba, or to Cuban nationals.” [59]

Even though Gentoo is being utilized instead of Debian, it would seem that both distributions would not be legally distributable to Cuba. Further follow-up discussion discusses that *Debian is not illegal* in Cuba, but the *knowing distribution* to Cuba is illegal. The discussion highlights the difficulty understanding the implications of embargoes on software and the extent to which (if at all) a distributor could face legal repercussions if that distributor knew that the software would be distributed to an embargoed country.

3.2.0.5 Policies of the Ecosystem

This category comprises licensing bugs related to the licensing policies of specific open source communities, *e.g.,* the Apache Software Foundation, Debian, the Eclipse Software Foundation, or Fedora. The bugs are originated from how a particular community defines the *freeness* of software, which can be stricter than utilizing a free/open source license, and how the community interprets licenses based upon their policies.

3.2.0.6 Community Guidelines

Description: This category relates to discussions about both the creation and modification of the guidelines of the ecosystem. These decisions determine the vetting of a software license's compatibility with community goals as well as manifests the perspective of a particular community towards free/open source development. Additionally, this category pertains discussion related to creating new licenses in order to comply with a community's guidelines.

Stakeholders: Community and lawyers.

Implications: Issues in this category have wide reaching repercussions and impact on a community. Unlike other licensing bugs, these reflect the collaborative effort within a community to enforce a particular licensing policy. The community needs to protect itself from legal liability of distributing potentially incompatible code, while also forging collaborations between the open source community and corporations. Thus, lawyers are also important in designing the guidelines to legally protect the community from potentially violating licensing. The agreed upon guidelines will have a long term impact in the community and will serve as a contract that specifies what licensing actions are appropriate and which are not. As a consequence, they have the potential to restrict the pool of software that can be reused/integrated into a new product.

Examples: We observe an issue with Eclipse's community restrictions on software licensing. To circumvent the restriction, end-users assume the burden to properly configure their environment to utilize that software. The developer states:

> "A number of projects would like to integrate with external libraries under licenses not compatible with the EPL (LGPL, ...). What can be done in order to help end-users come up with a working environment easily? Clearly, * The LGPL library must be installed from a source hosted outside Eclipse * A CQ must be filed indicating the dependency as "works-with" or "requires" But how can end-users or other clients be

instructed to get the complete environment going?" [52]

This shows the difficulty in meeting the needs of developers, while also facilitating reuse. The community guidelines impose practices to impede the use of any restrictively-licensed (*e.g.,* LGPL, GPL, etc.) software. The community belief is that this mitigates licensing issues/risks. However, the example also demonstrates that risk-mitigation may come at the cost of more difficulties for end-users.

3.2.0.7 Freeness: can I reuse it?

Description: Some of the most important questions to ask about a given software system are: What is its license? Is this license giving me the rights to incorporate the software into my own product? For example, the Free Software Foundation (FSF) and Open Source Initiative (OSI) define what, in their opinion, free and open source software (respectively) is. Furthermore, the FSF has specific guidelines on whether software with various licenses can be combined/derived along software licenses under the FSF licenses. Debian's DSFG indicates what are the characteristics of licenses for software to be integrated in Debian distributions.

This category affects the integration of many types of artifacts, such as source code, images (icons, logos), databases, text files, etc. Ultimately, the creation and enactment of similar guidelines by anybody developing software (whether free/open source or not) provides an opportunity for an up-front discussion on what legal risks an organization is willing to take, and, as a consequence, the pool of artifacts that can be reused.

Stakeholders: Integrators, package maintainers, & distributors.

Implications: In the case of Debian and Fedora, software with non-free licenses cannot be included in the main distribution and must be distributed separately. We observe one type of discussion relating to the freeness of free/open source licenses, in particular the GPL, QPL, AGPL, and Artistic License. Additionally, it is important that the entire system meets the community guidelines, including the needed dependencies. However, source code and binaries are not the only artifacts evaluated. Since IP clearance/evaluation extends to all bundled artifacts (not only source code and binaries), a *non-free* image or font could prevent the distribution of the software. Last, but not least, the community discusses specific clauses concerning how to deal with "Freeness" requirements.

Examples: We observe in RedHat's Bugzilla tracker that a license of *mpage* prevents source code modification, which makes the license non-free. The reporter of the bug states in the first two comments on the issue tracker:

> "mpage included non-free code, Please see Debuan [sic] bug number "805370" details. I think that this package be affected by debian bug number "805370"...Blocking FE-Legal, This is license problem." [53]

In summary, the statement above highlights the impact of the licenses in terms of one's ability to re-distribute the software.

We also observed that the "Choice of Venue" clause, which stipulates the location for a lawsuit regarding license infringement, sparked an extensive debate, since it imposes an additional restriction on developers reusing the software. At the same time, the absence of such a clause could penalize the original developer.

During the debate regarding which side can be discriminated against, one developer cited the opposing situation to a frivolous lawsuit by the original author by stating:

> "Whereas the alternative may be that licensors are unable to afford the enforcement of their license. Would you prefer to discriminate against them?" [58]

While the goal is to prevent disenfranchising someone due to his or her financial status, the "Choice of Venue" clause is inherently difficult to evaluate as it is orthogonal to this factor.

3.2.0.8 Potential License Violations

This section discusses the typical licensing bugs arising because licensing clauses have been violated, *e.g.,* due to integration with components containing incompatible licenses.

3.2.0.9 Compatibility between Licenses

Description: This category is related to issues relating to reusing software with different licenses. These issues arise when a developer utilized dependencies or reused source code that is not compatible with either the declared license, or with the license of other reused components. We observe incompatibilities between the license of dependencies to the systems reusing (or seeking to reuse) them. Additionally, a dependent artifact may change its licensing, possibly introducing a licensing incompatibility. These bugs also discuss the compatibility of licenses to understand under what condition software under two licenses can be used together (if all). These compatibility issues may also be related to a community's standards, *e.g.,* Apache Software Foundation requiring the Apache Software License for submitted contributions.

Stakeholders: Integrators, package maintainers, and distributors.

Implications: These incompatibility issues impact the ability to distribute the software, since at least one license of reused software is being violated from the incompatibility. These issues will either stall a patch or prevent a submitted system from being distributed within a community. These bugs can require substantial effort to fix, depending on the available replacement options for the violating code or the ability to migrate the software's license in order to be compatible with the reused code. To address these issues, the original developers need to remove the incompatible code/binary and find an alternative implementation that is licensed under a compatible license. In the case of a community's standard, it can also be problematic, since migrating the system's license can introduce other violations between dependencies or violate the community guidelines. For example, a project belonging to the Apache Software Foundation cannot migrate towards the GPL license. Additionally, a license change can prevent newer releases of a certain library from being integrated and forcing developers to rollback to a compliant release of that library. Thus, these issues are not always trivial to fix and can cause a longer delay for a contribution to be accepted.

Examples: In an issue on GitHub, a developer comments:

> "As mentioned in twbs/bootstrap#2054 as well as here, Bootstrap can not be used within any GPLv2 project as the Apache License 2.0 is incompatible. This just needs to be re-licensed under the GPLv3 to resolve this issue."[50]

The project's owner acknowledges that the license will be updated during the current rewriting of part of the code base. Interestingly, another respondent follows up saying that the compatibility is based on the Free Software Foundation's interpretation and the Apache Software Foundation differs; however, the assertion is either misinterpreted or outdated as the Apache Software Foundation does not assert compatibility with GPL-2.0 [74].

Similarly, we observe an issue in the Apache mailing list stating:

> "Be advised that at least 5 Apache projects appear to depend on a third party library known as "greenmail" which advertises itself in may [sic] places as being licensed under the ASL 2.0, however most of the greemail [sic] source files contain headers claiming LGPL" [48]

The issue relates to a dependency for unit test code (greenmail) and whether it is acceptable under Apache's licensing guidelines. The community discussed whether the dependency is acceptable, since the code is *not bundled and not distributed.* However, the issue was resolved in a new release of greenmail updating its licensing to ensure that all files were licensed under Apache-2.0.

3.2.0.10 Other Types of License Violations

Description: This category addresses issues that are considered license violations but do not fall into the prior category. These bugs are serious in nature and usually require removing or rewriting code. Otherwise, it might not be possible to continue distributing the product. These license violations can also be related to several other categories in our catalog, *e.g.,* incorrect licensing of derivative work or incorrect interpretations of licensing clauses. License violations can apply beyond software licensing with potential violations of the End User License Agreement (EULA).

Stakeholders: Distributors, integrators, and developers.

Implications: License violations prevent software, whether it is a patch or an entire system, from being distributed. By violating the license, it amounts to illegally distributing copyrighted material. These violations can only be addressed by the original developer, since the infringing code must be removed, or the software's license needs to be changed (if that addresses the violation). In fact, a Munich district court's injunction prevented the distribution of Fortinet Ltd's software until complying with the GPL [76]. Thus, it can directly impact a person or company's ability to distribute their project, and it could result in other civil penalties. Therefore, these licensing bugs are among the most severe as they directly impact software distribution and can involve a large technical effort to remedy the violation.

Examples: We observed a severe violation where it appears that a developer stole code and removed the original author from the copyright statement of the file(s). The author posts to the GitHub issue tracker stating:

> "Right now you're in violation of the MIT license that ember-tools uses. http://opensource.org/licenses/M
> I am the copyright holder of the code you've essentially stolen. You have not kept my name in the copyright holders."[1]

The issue demonstrates how violating the terms specified in the license, in this case the MIT license, results in the code being essentially stolen, since it is misappropriating the copyright holder by removing him from the copyright notice.

We observed an email thread where ICQ's EULA prohibits the connection of third-party software and a developer suggests removing the third-party applications from inclu-

[1]Quote anonymized given the sensitive nature to protect the identity.

sion in Fedora, stating:

> "...Thus, keeping in mind, that there is no way to use ICQ-related software w/o explicitly violating their license agreement, and there are many other open alternatives (and even proprietary systems, which permits 3rd party applications), I think that software, designed to work with explicit requirement to be connected to ICQ network, should be considered as unacceptable for inclusion into Fedora"[70]

The discussion regarding this draws into consideration whether such restrictions would then result in the software being non-free.

3.2.0.11 Non-Source Code Licensing

Non-source and non-binary licensing constitute an important aspect of license compliance that has not been addressed in prior works, which have focused predominantly on the licensing of source code or inferring the license of binaries. Tools to support software license compliance should also consider non-code and non-binary artifacts to properly evaluate license compliance.

3.2.0.12 Documentation Licensing

Description: Documentation, like source code, is also protected by copyright. Hence, it is necessary to license it. Free/open source licenses must make sure that the documentation can also be modified and redistributed. While software licenses were designed with source code in mind, we have observed that they are frequently used to license documentation. Although developers can utilize the GNU Free Documentation License [73] or a Creative Commons license [56], not all of the Creative Commons licenses may be considered free (*e.g.,* Debian indicated that the Creative Commons Attribution License version 1.0 is not compatible with the Debian Free Software Guidelines), which can cause issues within a community from distributing the documentation.
Stakeholders: Distributors.

Implications: These licensing bugs could prevent a package from being accepted for redistribution by a community, since the redistribution would either constitute distributing a copyrighted work or distributing an incompatibly licensed work (incompatible in the sense of community standards). Migrating the documentation's license may be the easiest way to address these types of bugs. However, we also observe a potential license violation for documentation, when the *source* of the documentation is not available. While the documentation itself is available, it may not be modifiable according to the terms of the software license.

Examples: We observe an issue where the documentation (licensed under the GPL) is provided as a set of HTML files. However, these files were automatically generated. The individual states:

> "...no source code is provided for the .html documentation files. The GPL is explicit on the definition of source code: 'The source code for a work means the preferred form of the work for making modifications to it.' The term 'source code' has a precise meaning within the GPL, which differs slightly from the everyday use of the term. The html files are not source code as the term is used in the GPL license, because the documentation is maintained in a different form, and converted to html. Html files, while human-readable, are extremely inconvenient to modify. They are certainly not the 'preferred form' for making modifications." [69]

Since the HTML was automatically generated, the developer's comments indicate that the software license would *require* the documentation's source, which is not provided. However, it is important to note that this example *prevents others* from redistributing the software. One developer noted this by responding:

> "I don't see how Trolltech are violating anything though, since they own the copyright. They just aren't being particularly useful, so we can't redistribute the offending html documents" [67]

Therefore, the copyright holder can distribute the material. However, the terms of the license cannot be satisfied, which prevents others from redistributing the documentation.

3.2.0.13 Font and Media Licensing

Description: We observed issues and discussions related to the licensing of fonts and media (*e.g.,* images and audio). In both cases, we observed issues related to redistribution. In the case of fonts, we observed potential violations to the font licensing as well as non-freeness of their licensing. Similar violations occurred for media, along with some confusion on the interpretation of software licensing in the context of media, *e.g.,* audio. Additionally (and expectedly), we observed discussions concerning copyrighted images that could not be re-distributed with software.

Stakeholders: Distributors.

Implications: We observed that the licensing of these artifacts could also impact the ability to distribute a system. Indeed, these non-source/non-binary artifacts are still subject to community guidelines compliance, which developers may not realize initially. In addition, the implications of software licensing is not as clear in the context of non-source or non-binary entities. For example, interpreting the concept of *source code* (*e.g.,* the GPL-3.0 defines it as "the preferred form of the work for making modifications to it" [13]) is less clear in the context of audio (*i.e.,* does the audio file itself qualify or does it require a different representation as well). To address these issues, non-free fonts or images might have to be removed, but this also requires developers to find replacements.

Examples: We observed issues related to media resources licensing and copyright images. In a discussion regarding the "source code" of audio licensed under the GPL-2.0, one developer noted:

> "Unless the creators of the podcast directly edit the MP3–which is rather unlikely–the MP3 is not the preferred form for modification and putting the MP3 under GPL

without releasing the raw audio files grants no rights at all. GPLing video has a similar problem." [60]

The issue demonstrates the difficulty of utilizing software licenses for media resources, since the creator may not be aware, as another developer indicates:

"It's a bug. If the original author puts a video under GPL and doesn't release the "source", you can't demand it. He's not bound by the GPL since he can't violate the copyright on his own work, so he has no obligation to give you anything."[61]

However, these bugs may also be due to the creators of the audio or video files being unaware of this consideration, since they do not store the intermediate files when finished.

3.2.0.14 Licensing Content

This category describes bugs related to how licenses are documented in software projects, and whether there have been some inconsistencies in doing that.

3.2.0.15 License Inconsistencies

Description: License inconsistencies occur when there is a mismatch between the documented license and the source code licensing. These licensing bugs may relate to the location of the licensing file, the usage of licensing-related macros and annotations (*e.g.,* in Ruby `.gemspec` files), licensing headers on the source code, or how multi-licensing is represented in the specification or documentation. Thus, the system's licensing may not cause a license incompatibility, but its content could be misrepresented or incomplete.

Stakeholders: Distributors and package maintainers.

Implications: These licensing bugs typically do not impact the acceptance of a package or patch (*i.e.,* it does not prevent IP clearance checks). However, they may stall the software distribution. In particular, certain communities may require the license to be in a spec file before the community can list and distribute the software. Also, they are typically issues of documentation (*i.e.,* errors of how to document) more than errors in licensing. These licensing bugs are relatively simple to fix, since the original developer(s) can add the proper annotation, move the license file, or add license headers without having to modify the actual system (*i.e.,* the source code itself does not require any change). Additionally, the developer typically receives feedback regarding the inconsistency during the contribution review process.

Examples: During a package review in RedHat's Bugzilla tracker, the reviewer evaluated the licensing and found that there was an inconsistency in the software's licensing and the spec file. The reviewer notes at the top of the package review:

> "To fix: Licensing is both MIT/BSD and GPLv2 and GPLv2+, latter two are missing in spec file and in %license."[54]

While these licensing bugs are relatively trivial to fix as the comment reported above suggests, it is important to properly enumerate the complete licensing from a package manager's perspective.

3.2.0.16 Other Intellectual Property Issues

This category does not pertain specifically to licenses but, rather, it is related to whether there have been issues in handling intellectual property in a software project, for example because of how the Contributor License Agreement has been defined, or because of patent or trademark implications.

3.2.0.17 Rights to Use a Contribution

Description: Many software systems, in order to clarify the provenance of their code have instituted the requirement that any contribution should be accompanied with a CTA or a CLA [132]. Contributor License Agreements (CLAs) and Copyright Transfer Agreements (CTAs) are important for intellectual property (IP) reviews. A CLA stipulates the terms of contribution (*e.g.,* the legal right to submit the code). We observed three different types of bugs in this category: (i) CLA creation, where the project has not established a CLA for contributors, (ii) CLA changes, where the CLA itself needs to be amended, (iii) missing CLA, where a contributor has not submitted a CLA. Alternatively to a CLA, CTAs transfer the ownership of the copyright from the original author (or current copyright holder) to another person(s) or legal entity.

Stakeholders: Package maintainers and integrators.

Implications: Projects that require CTAs/CLAs do it to reduce their legal risks. They require any contributor to (i) guarantee that they have the right of the contribution, and (ii) license or transfer ownership of the contribution to the project. If this does not happen, these projects will not merge contributions/patches. Some organizations, *e.g.,* the Apache Software Foundation and the Eclipse Software Foundation, have strong policies and workflows to include the submission and verification of CLAs in the contribution process. It is important to note that CLAs/CTAs are optional in the sense that an organization is not required to use them. However, it demonstrates that these open source communities would rather reject contributions than increase the legal risk of distributing code that may contain a license violation.

Examples: We observed a message posted to *appframework*'s issue trackers where a developer stated:

> "If you're interested in clean IP, you'll want a CLA before accepting any contributions – otherwise, you may encounter issues of people contributing code that is not theirs to contribute. For example, see the Dojo CLA. " [49]

The issue highlights the importance of having a CLA from the perspective of integrators of submitted patches. The issue poster states the concern that the IP could be compromised if the submitter of a patch does not have the right to distribute that code. Lastly, the poster provides a CLA example to aid the project owners.

3.2.0.18 Patent-Related Issues

Description: These bugs are related to the use of software patents owned by the licensor of the software. Software patents protect the ideas and inventions used in the source code and binaries, while copyright protects its expression. We observed issues related to patent-clause of licenses, in particular automatic license-termination in the event of a patent lawsuit. Alternatively, there are also discussions regarding the implications of patented artifacts.

Stakeholders: Lawyers and distributors.

Implications: Recently, licenses (*e.g.,* GPL-3.0 and Apache-2.0) have started to address particular issues related to patents, including their litigation. For example, engaging in patent litigation with work that is derivative or reuses source/binaries under these licenses will invalidate the license. However, companies such as IBM have written similar clauses related to automatic license-termination, which sparked further debate on the freeness of these license (*i.e.,* the ability for an organization to revoke a license at anytime adds a "restriction" on reuse). The automatic-termination and the debate on freeness may prevent the distribution of the software. In the first case, the software cannot be distributed if the license is revoked; in the latter case, the community's guidelines may prevent the distribution of the software.

Examples: In *debian-legal*, a question related to IBM's license-termination clauses emphasizes the difficulty of dealing with software patents:

> "It's not possible to know about what patents cover a work of software because: (a) the existence of a patent is secret until it's approved (b) software is just an abstract collection of energy – it's what it symbolizes that makes it tread on a patent, or not (c) patents are written in a fashion which does not make their applicability to a work of software immediately evident." [65]

The person explains that patented code would be problematic regardless of a termination clause within IBM's license, and suggests it should be acceptable to Debian. However, the message also demonstrates the difficulty of patented software, since the patent may be approved at a later date and the scope/applicability of the patent may require legal consultation.

3.2.0.19 Trademark

Description: According to the United States Copyright Office, "A trademark protects words, phrases, symbols, or designs identifying the source of the goods or services of one party and distinguishing them from those of others." [75]. Primarily, we observe the discussion related to trademarked names and logos, such as the Debian logo and package/bundle names.

Stakeholders: Distributors, Communities & Trademark Holders.

Implications: Companies use trademarks to protect their *brand*, since they prevent external entities from misrepresenting their products or organization. These restrictions can prove problematic within an community. Organizations want to provide a quality guarantee, but communities like Debian, debate the freeness of these restrictions (*i.e.,* preventing someone from using the same product name or logo). The concern is predominantly for the distributor that needs to ensure that trademarks are upheld. However, it also impacts the community, since the members must determine if these restrictions are in-line with their guidelines. The trademark holder can license the usage of trademarks and address these issues.

Examples: For example, Mozilla allows for their Firefox and Thunderbird trademarks to be used to identify *unmodified* official releases [78]. Organizations can impose different requirements regarding trademark usage, but it can be seen as a mechanism to guarantee a level of quality similar to the original system [93].

In *debian-legal*, there are several discussions related to freeness of trademarks. One commenter replied regarding trademarked names and distributed package names by stating:

> "Over the years there have been a large number of packages in the archive for software whose upstream has a trademark on their name, none of whom have granted open-ended licenses to use these trademarks. We nevertheless have always made a practice of using those names unmodified as package names and binary names, on the grounds that these are interfaces that are *not subject to trademark*. This is why, whereas RedHat ships packages of 'httpd', Debian has always had packages of 'apache' even though the Apache trademark license clearly states that modified versions of the software may not use the mark. The trademark license is only relevant if we're doing something that's in scope for trademark law in the first place!" [66]

The message highlights the policy of Debian to keep the names and suggesting that such a usage is analogous to the scope of trademark law. Interestingly, the message also shows how a company like RedHat takes a more cautious approach to prevent possibly infringing on trademark restrictions.

3.2.0.20 Licensing Semantics

This section discusses licensing bugs related to difficulty and confusion with the usage of dual licensing or understanding the implications of either a license or particular clauses of a license.

3.2.0.21 License/Clause Implications

Description: These licensing bugs relate to understanding the implications of the license as a whole or a particular clause of a license. These licensing bugs can relate to the discussion of license migrations, where developers discuss the implications of the license migration on their system and if the new license would have legal implications on the current state of the system (e.g., the license migration could result in a license violation based on licensing constraints from reused code or dependencies). These licensing bugs also discuss the implications that a clause, such as the *"or later"* clause of the GPL, on a system in the future.

Stakeholders: Developers and Integrators.

Implications: The litigious nature of licenses can make understanding the implications of certain licenses difficult. This can result in developers picking a license that does not meet their needs or expectations. Similarly, it can prevent developers from adopting or migrating towards certain licenses based on their uncertainty of the ramifications of such a decision (*e.g.,* whether the new license is compatible with the licenses of the software's dependencies or whether a license satisfies the business model of the developers). Additionally, it may impact integrators trying to reuse the software, since the original developers may be apprehensive to modifying the license from the lack of understanding. Conversely, integrators may also misunderstand the implications of a clause, which can inhibit their reuse of that software.

Examples: We observed developers discussing migrating towards GPL-2.0+ (*i.e.,* GPL "or later") in order to facilitate reuse in software licensed under GPL-3.0. However, the implications of this *"or later"* clause was also seen as potentially dangerous. One developer indicated the possible risk of not agreeing with the terms of a future version of the GPL. Although the protections and restrictions of such a license are unknown, the clause indicates that the software can be licensed under such terms. Another developer expresses apprehension towards this stating:

> "In fact, you should not trust any third party (even if it is FSF) about future modifications of the license which have not been taken into account by the copyright holders." [80]

The example demonstrates that the importance regarding the implications of a clause, which may appear harmless initially (as developers in the discussion initially seemed to open to the change) but could have more a serious long term impact.

3.3 Related Work

A substantial part of the research work carried out in the software licensing field focused on the definition of techniques and tools supporting the automatic identification of licenses in software products. For instance, the FOSSology project [107] exploited machine learning techniques to automatically classify licenses, thus, supporting the task of license identification for a given software. Tuunanen *et al.* [146] presented ASLA, a reverse engineering tool supporting the identification of software licenses with an accuracy of 89%. Afterwards, German *et al.* [106] proposed *Ninka*, an approach using sentence matching and currently representing the state-of-the-art, with a precision of 95%. Di Penta *et al.* [99] proposed an approach relying upon code search in order to identify the licensing of jars. Finally, German *et al.* [102] analyzed 523,930 archives in order to understand the impact the accuracy of identifying FOSS licenses when used in conjunction with proprietary licensing.

Previous work has also analyzed the usage, evolution and inconsistencies of open source licensing in software projects. Di Penta *et al.* [100] investigated the migration of licenses over the course of a project's lifetime. Their study suggests that licenses changed version and type during software evolution, but there were no generic patterns generalizable to the six analyzed FOSS projects. German *et al.* [105] analyzed 124 open source packages exploited by several applications to understand how developers deal with license incompatibilities. Based on this analysis, they built a model outlining when specific licenses are applicable and what are their advantages and disadvantages. Later, German *et al.* [103] presented an empirical study focused on the binary packages of the Fedora-12 Linux distribution aimed at (i) understanding if licenses declared in the packages were consistent with those present in the source code files, and (ii) detecting licensing issues derived by dependencies between packages. As a result of their investigation, German *et al.* [103] were able to find some licensing issues confirmed by Fedora. Manabe *et al.* [119] analyzed the changes in licenses of FreeBSD, OpenBSD, Eclipse, and ArgoUML, finding that each project had different evolution patterns. Vendome *et al.* [152] investigated license usage and changes in licensing to understand the rationale behind potential usage and changes. Vendome *et al.* [153] also investigated when developers pick a particular license or changes the license(s) and conducted a survey to understand the underlying perspective and rationale of developers with respect to choosing or changing their system's licensing. Almeida *et al.* conducted a survey to investigate the extent to which developers understand licenses [85]. License inconsistencies have also been analyzed in a study by German *et al.*, which considered code clones between Linux and two BSD distributions [104] as well as a study by Wu *et al*, which considered code clones in Debian 7.5 with inconsistent licensing, potentially representing license violations [154].

Most of the aforementioned work has focused on C/C++ and Java languages. However, few studies have focused on other languages, for instance, Mlouki *et al.* [125] investigated license violations and their evolution in 857 open source Android apps; 229 releases for 17 of the analyzed apps were found to have license issues, in particular, violations to the terms of open source licenses. Vendome *et al.* [151] found 14 different license exceptions (i.e., additional grants/restrictions specified by the copyright holders beyond the canonical license) in 298 files from open source projects developed in six programming languages; although license exceptions are not prevalent in open source projects, they may introduce license bugs since they modify the canonical versions of open source licenses.

The study presented in this paper is orthogonal (but complementary) to prior work. Indeed, we aim at defining a *detailed catalog of licensing bugs faced by developers*, with the goal of characterizing them, studying their implications, and how they are fixed. **To the best of our knowledge, this is the first work systematically investigating the nature and implications of licensing bugs**.

3.4 Discussion

The analysis of the commit messages and issue tracker discussions highlighted that the Copyright laws are complex, and frequently have no simple answer. Some issues are relatively well understood (such as copying code from one application) but others are not. We observed that in some free/open source projects there are discussions in terms of whether an action has the potential to violate copyright; usually, these discussions are among developers, who do not necessarily have the proper understanding of copyright law, and usually lack professional legal advice. The main implication is that there exists an overall lack of training or education about software licensing, and developers need to be better trained on copyright law in order to have a better understanding of the potential risks. Large organizations sponsoring free/open source projects should have legal advice available to minimize such risks. Similarly, the discussions and (mis)interpretations illustrate developers would greatly benefit from more support by (semi)automated license analysis.

Additionally, communities acknowledge these difficulties relating to copyright and define guidelines to reduce the legal risk of distributing software. Debian achieves this through the DFSG, which vets potential licenses according to freeness; however, the Apache Foundation asserts that contributions must all be licensed under the Apache license. Free/open source projects also contain a review process and can use CLAs/CTAs as an IP assurance mechanism. Developers should carefully consult the community practices prior to contributing to decrease the potential licensing-related impediments when contributing. Future licensing recommenders should incorporate community's expectations/conventions.

Different international laws impact free/open source code reuse. While, in principle, software can be made available for download everywhere, in some cases, reuse can have different legal implications in different countries, or in some cases, might not even be permitted outside a given country.

Finally, a software project may comprise artifacts beyond source code, including documentation or media. Their copyright and license must also be accounted from a compliance perspective. Also, importantly, if some of these artifacts (*e.g.,* documentation) are automatically generated from a source code, the latter shall be included to consider the whole software distribution to be free/open source.

3.5 Bibliographical Notes

The publication supporting the content described in this Chapter was performed in collaboration with other members of the SEMERU group at William and Mary and researchers from the Universidad de los Andes, University of Sannio, University of Lugano, and the University of Victoria:

- **Vendome, C.**, Germán, D., Di Penta, M., Bavota, G., Linarres-Vásquez, M., and Poshyvanyk, D., "To Distribute or Not Distribute: Why Licensing Bugs Matter", in Proceedings of 40th ACM/IEEE International Conference on Software Engineering (ICSE'18), Gothenburg, Sweden, May 27 – June 3, 2018, to appear 12 pages.

Chapter 4

Machine Learning-Based Detection of Open Source License Exceptions

As described by the License Proliferation Report by the Open Source Initiative [145] and reported by empirical studies from the research community [153, 152, 135, 137, 136, 105, 103], proliferation of free/open source licenses implies that (i) it is hard for developers to choose the right license for their projects and goals; (ii) there are incompatibilities between some licenses that might be a threat for developers' goals; and (iii) reuse of free/open source software can lead to projects with multi-license distributions.

One partial solution to the aforementioned issues is the definition and usage of license exceptions that, when attached to a license, modify it without changing the text of the license itself [105]. For example, the MySQL database management system faced a challenging problem: It needed to stop commercial companies from reusing the MySQL connectors library (required to connect to the database) while still allowing other free/open source projects to continue using it. The first issue was easily addressed by using the GPL (since it would require those companies to also release their products under the GPL, unless they bought a commercial license). However, doing so would also stop projects that were not GPL licensed (such as PHP) to continue connecting to the database. The solution MySQL AB chose was to add an exception to the GPL (the MySQL FOSS License Exception) that allows software under certain free/open source licenses (including the PHP license) to use the connector libraries, effectively altering the scope and impact of the GPL on such software projects [128].

Licensing usage in free/open source software has been investigated in several studies, mainly focused on identifying the prevalence and adoption of free/open source licenses and the developers' rationale under their licensing choices [135, 137, 136, 152, 153].

However, no previous effort has been devoted to analyze the prevalence/adoption of license exceptions. Given the large volume of free/open source projects available in forges (not only for reuse but also for direct usage), license exceptions might result in legal issues when developers/users are neither knowledgeable of the exceptions declared in the free/open source software nor of the implications. While reliable tools for license identification and classification exist [107, 146, 99, 102], and some of these tools can be used to check for licensing inconsistencies [103], no tool is available to support the identification and analysis of license exceptions.

In this Chapter, we aim to address two main questions:

- *Are the exceptions to free/open source licenses used by the community, which exceptions are used, and to what extent?*
- *Can we automatically detect license exceptions with a high precision and recall?*

We address these questions by first performing a large scale mining-based study in which we analyze the phenomenon of license exceptions from both a quantitative and a qualitative perspectives. In particular, we analyzed the source code of 51,754 projects written in six different programming languages (Ruby, Javascript, Python, C, C++, & C#) and hosted on GitHub, looking for usages of the license exceptions reported in the SPDX list [143]. By using defined heuristics, we found files identified as potentially containing license exceptions. Then, we manually inspected the files to (i) remove false positives, and (ii) categorize the exceptions according to their purpose.

Having assessed the magnitude of the license exception phenomenon used in free/open source projects, we propose an approach aimed at automatically identifying exceptions (if any) declared in a license.

Given a licensing statement, our approach exploits the words in its text as predictor variables for a machine learner that classifies the type of exception (categorical dependent variable) reported in the statement or marks it as "not reporting exceptions". We evaluated the precision and recall of our approach using Decision Trees, Naive Bayes, Random Forest, and Support Vector Machine (SVM). SVM outperformed the other classifiers, achieving an F-Measure higher than 95%. Since this is the first work to tackle the identification of license exceptions, we compared our approach with a baseline using template-matching techniques to identify license exceptions, showing the superiority of the SVM-based solution.

4.0.1 The Impact of License Exceptions

A license exception, when attached to a license, changes its meaning and implication. Specifically, an exception can change the requirements of the license (expand them or narrow them) and/or the grants of the license (again, expand them or narrow them). A license with an exception effectively becomes another license. However, license identification tools have focused on identifying the licenses, and while they identify some exceptions, there has been no research in this area.

A recent discussion in the SPDX mailing list (SPDX, the Standard Package Description Language, is intended to create a standard format to document licensing information in software) focused on a relatively unknown exception created by Sun Microsystems and used along with variants of the BSD and the MIT license. This exception [39] states (emphasis is ours) [24]:

> You acknowledge that **this software is not** designed, **licensed** or intended **for use in** the design, construction, operation or maintenance of **any nuclear facility**.

One of the outcomes of this discussion is that this exception appears to restrict the original license to non-nuclear facilities ("is not [...] licensed [...] for use [...] any nuclear facility"). While the original license (BSD-like) is open source, the addition of this exception potentially turns it into non-open source because it appears to contravene clause 6 of the Open Source Definition that requires a license not to discriminate against fields of Endeavor [26].

License identification tools, Ninka [106] and Fossology [107], report licenses that they identify, but are oblivious to information that they do not recognize as a license. For instance, Fossology identifies this Sun license—**including the exception**—as *BSD-style*. This implies that the license is open source, even though the exception potentially makes it non-open source.

We searched Debian source files to see if this license was being used in open source systems. We identified two main variations of this exception, and we found it in two packages of Debian. If this license is considered non-open source, that contravenes the Debian Guidelines and these packages need to be reviewed for their inclusion in Debian (we have reported this issue to Debian).

Other exceptions, such as the MySQL FOSS License Exception, and the Java CLASSPATH Exception [141] soften the restrictions of the GPL regarding derivative works. A library that is licensed under the GPL requires any derivative works that use it to be also licensed under the GPL. The MySQL FOSS License Exception when attached to the GPL allows the creation of derivative works that link to a library without also having to be licensed under the GPL as long as the derivative work is licensed under one of the free/open source licenses the exception lists. The Java CLASSPATH Exception is broader and allows the creation of derivative works under any license (including commercial) that link to libraries (licensed under the GPL plus the Java CLASSPATH Exception). If a library was licensed under the GPL plus either exception, but the exception is not identified, potential users of the library would not use it because they would not be willing to license their software under the GPL (something that the exception allows them to do).

4.1 Empirical Investigation on License Exceptions in GitHub Projects

The *goal* of this study is to investigate the phenomenon of license exceptions in free/open source projects hosted on GitHub. The *purpose* is to understand the relevance of license exception usage, identify what kinds of exceptions are being used, and understand implications of free/open source license exceptions. The *perspective* is of researchers interested in supporting developers with respect to license compliance and verification. The *context* consists of the change history of 17,984 Ruby, 14,161 JavaScript, 9,349 Python, 4,671 C, 3,690 C++, and 1,902 C# free/open source projects mined from GitHub.

4.1.1 Research Questions (RQs)

We aim at answering the following research question:

- **RQ$_1$:** *How prevalent are license exceptions in free/open source systems?* This research question analyzes the prevalence of different types of exceptions to free/open source licenses for projects hosted in GitHub and written in the six programming languages we considered. The goal is to understand license exceptions in practice, since they have not been investigated in prior studies. Besides *quantitatively* measuring the diffusion of different types of license exceptions, we also contacted the developers of the systems in which we identified exceptions to understand whether they are aware of the license exceptions. Additionally, we *qualitatively* discuss prominent cases that we found in order to better understand the context in which license exceptions are used.

4.1.2 Data Extraction Process

In order to identify license exceptions, we analyzed the commit history of 51,754 projects hosted on GitHub and developed in six of the most popular programming languages on GitHub [1]. We leveraged the project metadata to filter the repositories such that they had at least one star, watcher, or fork and were not themselves a fork (*i.e.,* removing abandoned or personal repositories, and preventing duplication). We locally cloned the 51,754 project repositories to perform our analysis.

For each file f_i in the locally cloned repositories, we used *Ninka* [106] to extract a *comment file* C_{f_i} containing all source code comments in f_i (and, therefore, the license header with the exception text, if any). Then, we defined a set of heuristics to identify (candidate) license exceptions in each comment $c_j \in C_{f_i}$. The authors defined these heuristics by manually inspecting the description of the known/accepted license exceptions listed in SPDX[142]. In particular, we looked for sentences and keywords representing "markers" for the presence of a (specific) license exception. In the end, we defined the following heuristics, assuming that a comment c_j reports a license exception le_k if:

H$_1$ c_j contains the exact text (*i.e.,* definition) of a license exception le_k;

H$_2$ c_j contains the le_k's exception name (e.g., "autoconf" for the *Autoconf Exception*) and the token "exception";

H$_3$ c_j contains the string "as a special exception", a quite common pattern across the exceptions listed in SPDX.

It is important to note that these three heuristics are purposefully designed to address recall of license exceptions in order to identify the possible presence of license exceptions that were not listed by SPDX. **H**$_3$ was designed such that it might be able to identify license exceptions not reported in SPDX. Also, as with any heuristic-based approach, our heuristics can lead to the identification of false positives. We deal with such limitations by manually analyzing every comment identified by our heuristics as reporting a license exception. In particular, our manual analysis (i) validated the presence of the license exception (*i.e.,* discarded false positives) and (ii) assigned a license exception name. If a comment reported a license exception without a known name, we assigned a custom name to the exception. Overall, our heuristics identified 728 files reporting candidate license exceptions for $\mathbf{RQ}_1$; then, we manually analyzed the files, getting 298 (40.9%) files with license exceptions (true positives). As previously stated, this true positive ratio is expected, since the heuristics were designed to capture exceptions that were not included by SPDX (while this reduced precision, we minimized the impact on our findings through the manual validation). From the 298 files, we identified fourteen unique exception types, six of which are not documented/reported in the SPDX list.

To answer $\mathbf{RQ}_1$, we report the diffusion of the different exceptions in the mined repositories when considering them together and separately by different languages. It is important to note that the results report only systems with license exceptions and not the diffusion of licenses (*i.e.,* we do not consider all licensing of the 51,754 projects, but the subset that are licensed and have a license exception). Therefore, after the manual validation, we obtained a set E of tuples $E_j = \langle f_i, except, lang \rangle$ with *except* being the license exception name, and *lang* the programming language used in f_i.

4.1.3 Results for $\mathbf{RQ}_1$: How prevalent are license exceptions in free/open source systems?

Table 4.1 shows the number of files reporting each of the 14 identified exception types across five programming languages (JavaScript is omitted, since we did not identify any exception in projects written in this language) as well as the number of projects (in parenthesis) containing each exception type.

4.1.3.1 Diffusion of different license exceptions

The *OpenSSL Exception* was the most prevalent, with 89 files having the exception across two systems. The *OpenSSL Exception* facilitates linking the licensed code to OpenSSL or

Table 4.1: Frequencies of license exceptions by language at file and system (in parenthesis) granularity.

Exception	**Ruby**	**Py.**	**C**	**C++**	**C#**	**Total**
Autoconf	11 **(3)**	20 **(7)**	11 **(1)**	30 **(3)**	0 **(0)**	72 **(14)**
Libtool	3 **(1)**	0 **(0)**	2 **(1)**	3 **(2)**	0 **(0)**	8 **(4)**
dh-Make	0 **(0)**	3 **(3)**	2 **(1)**	0 **(0)**	0 **(0)**	5 **(4)**
TexInfo	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	1 **(1)**
RACC	22 **(20)**	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	23 **(21)**
Bison	6 **(2)**	0 **(0)**	0 **(0)**	2 **(1)**	0 **(0)**	8 **(3)**
Nokia QT LGPL	0 **(0)**	0 **(0)**	0 **(0)**	2 **(1)**	0 **(0)**	2 **(1)**
Nokia QT GPL	0 **(0)**	0 **(0)**	0 **(0)**	49 **(1)**	0 **(0)**	49 **(1)**
Classpath	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	10 **(1)**	10 **(1)**
GUILE	0 **(0)**	2 **(2)**	0 **(0)**	0 **(0)**	0 **(0)**	2 **(2)**
Rails usage	19 **(18)**	0 **(0)**	0 **(0)**	0 **(0)**	0 **(0)**	19 **(18)**
MIF	0 **(0)**	0 **(0)**	8 **(1)**	1 **(1)**	0 **(0)**	9 **(2)**
OpenSSL	0 **(0)**	88 **(1)**	0 **(0)**	1 **(1)**	0 **(0)**	89 **(2)**
WxWin. Lib. 3.1	0 **(0)**	1 **(1)**	0 **(0)**	0 **(0)**	0 **(0)**	1 **(1)**
Total Files	62 **(22)**	115 **(13)**	23 **(2)**	88 **(7)**	10 **(1)**	298 **(45)**

derivative work that maintain the OpenSSL licensing terms. The *Autoconf Exception* was the second most prevalent (72 files containing the exception). This particular exception removes the copyleft requirement of the GPL when it is being used with a configuration script generated by Autoconf. The diffusion of the two exceptions above is not surprising, as OpenSSL is a widely diffused library, whereas Autoconf is a popular (and language independent) tool to generate configure scripts.

The *Nokia GPL Exception v1.3*, created to govern the redistribution of the Nokia Qt library, was the third most prevalent (49 files); however, all files reporting it were from the same project, *qtablet*. These 49 files were part of the *qtanimation framework-1.0-opensource*, which is a third party library for cross-platform software development in C++.

The *RACC Exception* (23 files) and *Rails Exception* (19 files) were the fourth and fifth most prevalent exception types, respectively. The similarity in frequency is due to these files being components of the same reused library. Therefore, these two exceptions were often found in tandem in the files `parser.rb` and `format.rb` belonging to *Action Mailer* [3], which facilitates sending and receiving emails in Rails applications. However, `parser.rb` was also utilized independently of `format.rb`. Racc is a parser generator for Ruby and the *RACC Exception* excludes the parsers that are generated by Racc from being licensed under the Ruby license (not yet approved by the Open Source Initiative). The *Rails Exception* allows for the usage of a MIT-like alternative license when the source code is used with the official Rails or systems built upon the official Rails.

The *Classpath Exception* was the sixth most prevalent (ten files in a single system). It allows for linking a library to independent modules with requiring the generated binary from being licensed under the terms of the GPL.

The *Macros and Inline Functions Exception* (*MIF Exception*), which allows unrestricted reuse of executable that utilize macros, inline functions, or instantiate a template from the file containing the *MIF Exception*, was the seventh most prevalent with nine files containing the exception.

The *Bison Exception* and *Libtools Exception* were found in eight files each. The *Bison Exception* allows for unrestricted reuse of the Bison skeleton as long as the system is functionally different (*i.e.,* not a parser generator), while the *Libtools Exception* allows for unrestricted distribution of the file if it belongs to a system built by Libtools. Subsequently, we observed *dh-make Exception*, which resembles the *RACC Exception* differing in that it applies to dh-make output files instead of Racc output files, and occurs in five files.

The *Nokia Qt LGPL Exception* and *GUILE exception* tied with two files. The *Nokia Qt LGPL Exception allows* for un-restricted reuse of binary code that (i) utilizes only unmodified header files, modified code impacting numeric parameters, data structure layout, or (ii) the modification adheres to the MIF Exception, and (iii) adheres to the LGPL's Section 6 (facilitating reuse of the work as a library). The *GUILE Exception* is an exception for the executable generated by linking GUILE Library to other source files to be exempt from the terms of the GPL (*i.e.,* the generated binary does need to be releases under the GPL). Finally, there was one file with the *TeX Exception*, which excludes LaTeX files generated by *texinfo* from being licensed under GPL, and one file with the *WxWindows Library Exception 3.1*, allowing for unrestricted reuse of binary code based on the library that contains this particular exception instead of enforcing the terms of the GPL on the binary.

4.1.3.2 Distribution of license exceptions across programming languages

For C, we observed 23 exception instances made up of four exception types. The *Autoconf Exception* was most prevalent in C (11 files). The second most prevalent was the *MIF Exception* (eight files), while both the *dh-make Exception* and the *Libtool Exception* were present in two files.

For C++, we had the greatest variability with eight different types of exceptions and the highest overall number of files reporting exceptions (88). The most prevalent was the *Nokia GPL Exception v1.3* with 49 files from the reused Qt Animation Framework. The *Autoconf Exception* was second most prevalent in C++ (30 files), followed by the *Libtools Exception* (three files). The *Nokia Qt LGPL Exception* and *Bison Exception* tied for fourth (two files), while the *OpenSSL Exception* and *MIF Exception* tied for fifth each with one file.

For C#, we only observed the *Classpath Exception* in ten files and only from the system *Chefrate*. The C# files were reused components for the system's Png Encoder within the cross-platform and cross-browser API. Interestingly, the license header of these files also indicate that they were translated from Java to C# and thus inherited the *Classpath Exception* from the original Java implementation.

For Python, we observed five different types of exceptions resulting in 115 license exceptions. The *OpenSSL Exception* was most prevalent with 88 files containing the exception. The second was the *Autoconf Exception* with 20 files, while *dh-make Exception* was the third (three files). Additionally, we found the *GUILE Exception* once in two different projects, and we observed one file with the *WxWindows Library Exception v3.1* and one with the *RACC Exception*. The latter was the Ruby `parser.rb` file nested under a directory of external libraries.

For Ruby, we found six different types of exceptions across 62 different files. The *RACC Exception* was most prevalent license exception (22 files) and was closely followed by the *Rails Exception* (19 files). The *Autoconf Exception* was the third most prevalent exception with 11 files. The fourth was the *Bison Exception* (six files). The *Libtool Exception* had three files containing the license, while the *TeX Exception* was only attributed to a single file.

It should be noted that the *Autoconf Exception* was found in systems written in four languages with a relatively high prevalence (*i.e.,* top-3 across all four languages). Also, it seems that the programming language may influence the types of found exceptions. Indeed, the *RACC Exception* and *Rails Exception* are inherently coupled to Ruby files, which explains their isolation to Ruby (although a Ruby file did contain it in a python project). Similarly, Nokia's Qt Framework supports C++ development and contains an extensive API, impacting both the frequency of the exceptions when the libraries are reused and the isolation to C++.

4.1.4 An Initial Discussion and Learned Lessons

In this section, we first present feedback collected by surveying developers. Then, we present a manual categorization of license exceptions by similar features or properties.

4.1.4.1 Developer Awareness of License Exceptions

After identifying the license exceptions, we contacted the developers of the systems and asked them if they were aware of the license exceptions and if they understood the exception text (we provided paths to the files with exceptions and the exception name to the developers). We received feedback from seven developers contributing to five of the 45 projects reporting license exceptions. While the low response rate limits the ability to draw conclusions from the developer survey, the responses are still useful to understand the perspective from at least a subset of developers.

Interestingly, five developers were unaware of the license exceptions. One respondent thanked us for bringing the license exception to his attention and he expressed his intention to fix the licensing statement. This case is particularly interesting since it demonstrates the difficulty that developers may have tracing the licensing constraints of third-party code and reinforces the need for an automated tool to support the identification of license exceptions.

Additionally, we asked the developers whether they were able to easily understand the particular license exception.

While only two respondents replied that they do not understand (and one indicated that licensing is troubling in general), three respondents expressed their understanding and a potential ambivalence regarding their understanding (*i.e.,* they indicated they were uncertain, but provided their interpretation). The responses suggest the difficulty that developers have when licensing extends to license exceptions, even in cases of more straightforward exceptions (*e.g.,* the *Autoconf Exception*). Additionally, it demonstrates that certain developers would benefit from licensing tools that provide more contextualized licensing analysis, especially when exceptions are present.

The developers' responses demonstrate that license exceptions may not be easily identified within reused third-party code. Additionally, the implications of the license exceptions may not be easy for developers to truly understand. While the sample is small and not generalizable, it does suggest that there are developers, such as package managers, who would benefit from tools to identify license exceptions and determine license compatibility.

4.1.4.2 Categorization of License Exceptions

Based on their purpose, we can classify the exceptions found in our study into three major categories.

Added by a third party and applicable to reused components embedded into the client software. In this category, we found the *Autoconf, Libtool, dh-Make, TexInfo,* *RACC,* and *Bison Exceptions*. In all these cases, exceptions were found in source code that has been generated by another tool (the tools have the same name as the exception and they are licensed under the GPL). The goal of these exceptions is to clarify that, even if the generated output might contain copies of GPL licensed-software, the GPL license does not affect the software that is using the generated code. For example, Bison, a well-known parser generator, embeds into its output source of Bison itself. Since it is licensed as GPL, and the parser must be compiled within the client program, this would require any client program also to be GPL. The *Bison Exception* removes this requirement: *As a special exception, you may create a larger work that contains part or all of the Bison parser skeleton and distribute that work under terms of your choice, so long as that work isn't itself a parser generator using the skeleton or a modified version thereof as a parser skeleton ...*

In these cases, the text of the exception is embedded into the code generated by the tool and is not added by the authors of the software where they have been found. This is the main reason why these license exceptions are the most prevalent.

Added to clarify or expand use of the software. These exceptions are used as part of the license of the software where it is found and has been explicitly added by the author of the software. This includes the *Nokia QT LGPL/GPL, WxWindows Lib. 3.1, Classpath, GUILE, Rails*, and the *MIF Exceptions.* In these cases, the owner of the product is using the exception to either modify the GPL or to clarify when the terms of the GPL do not apply. For example, the goal of the *ClassPath* and the *GUILE Exceptions* is to indicate that anybody is free to link to the (unmodified) library without having to release the code also as GPL.

The *QT Exception* is more limiting, since it only allows the linking with software that is licensed under specific free/open source licenses. The *MIF Exception* (Macros and Inline Functions Exception) clarifies that reusing templates and inlined macros (creating a copy of the original source code as part of the template and functions instantiation) is not a violation of the terms of the GPL:

[...] Specifically, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other files to produce an executable, this file does not by itself cause the resulting executable to be covered by the GNU General Public License. [...]

Added to allow linking to a library under a license that is not GPL compatible. This category contains only one exception: *OpenSSL*. OpenSSL is considered to be the best library in its kind, but its license is not compatible with the GPL. The exception allows explicitly the authors of GPL code to link OpenSSL even thought the license of OpenSSL is not compatible with the GPL:

[...] the copyright holders give permission to link the code of portions of this program with the OpenSSL library under certain conditions as described in each individual source file, and distribute linked combinations including the two [...].

Given the importance of OpenSSL, its prevalence is not surprising.

4.1.5 Threats to Validity

Threats to *construct validity* relate to the relationship between theory and observation, and can be mainly due to imprecision when detecting license exceptions with the textual heuristics. To mitigate this threat, we manually removed the false positives from the automatically identified candidate exceptions. In that sense, the results we report are based on true positives only. However, it is possible that some license exceptions were not detected by the heuristics.

Threats to *external validity* relate to the ability to generalize the results from the study, and we do not assert that the results in this paper are representative of the open source community. We only analyzed projects written in Ruby, JavaScript, Python, C, C++, and C#. Other languages and forges as well as commercial systems may exhibit different frequencies in terms of license exceptions. However, GitHub is the most popular forge with a large number of public repositories. Developers in other languages and utilizing other forges may have other perspectives related to license exceptions.

Threats to *internal validity* relate to internal, confounding factors that would bias the results of our study. We selected all projects from GitHub meeting the filtering criteria; therefore, we did not have a bias while selecting projects from a specific domain; however, because of time considerations we focused only on six programming languages. In the future work, we aim to expand our analysis to other programming languages, forges, and specifically investigate libraries.

4.2 Using Machine Learning to Identify License Exceptions

The study in Section 4.1 suggested that license exception identification is a cumbersome task as it requires developers to sift through a high number of potential false positives when using heuristics. Additionally, feedback from the developers suggests that an automated approach would be useful, especially for package maintainers. Automated approaches for text categorization/classification often rely on supervised learning to derive a model from labeled data (*i.e.,* text with labels) that can be used to categorize unseen data [133]. Text categorization has been successfully used in software engineering tasks such as software categorization [147, 121, 118], defect prediction [122], and developer recommendation [86, 117]. However, the task of automated classification of software licenses and license exceptions has not been solved using a machine learning-based approach. As a matter of fact, while tools to identify licenses exist [107, 146, 99, 106], no technique to automatically classify license exceptions has been proposed so far.

As we showed in our first study, keyword-based identification of license exceptions is highly prone to false positives, which suggests that "smarter" approaches should be used. One first option could be to use template-based identification (similar to Ninka [106]), or to use predefined queries to detect the license exceptions within a software license text. Our hypothesis here is that text categorization techniques can be used to detect license exceptions by ensuring higher accuracy with respect to techniques based on template-matching. Therefore, to assist developers in automatically identifying license exceptions, we implemented a text categorization approach. To validate the accuracy of the approach when classifying license exceptions automatically, we conducted an empirical study in which we compare the accuracy of supervised classifiers against a baseline representative of template-matching techniques.

More formally, the *goal* of this study is to evaluate a machine learning-based approach that we devised for license exception identification and compare it against a template-matching baseline for license exception identification. The *perspective* is of practitioners interested in ensuring license compliance of their systems, and the *context* consists of real license exceptions from our preliminary study and a synthetic dataset covering all license exception types.

4.2.1 Research Questions (RQs)

We aim at answering the following two research questions:

- **RQ$_2$:** *What classifier provides the best accuracy for license exception identification relying on machine learning?* We compare the performances of different classifiers for identifying license exceptions in licenses text.

- **RQ$_3$:** *Can our machine learning-based approach beat a baseline approach matching the license exception text?* RQ$_3$ aims at understanding whether a machine learning-based approach provides sufficient benefits to solve the problem of license exception identification. Thus, we compare it with a baseline approach (BL) that searches for the license exception text in the licensing statement.

4.2.2 Dataset Construction

In our first study, we did not find real examples for all the types of exceptions listed in the SPDX list [143]. Also, in some cases, we only found very few instances for a given exception (*e.g.,* the *Texinfo Exception* only had a single instance in our dataset). To train our machine learning (ML) algorithms by avoiding the "class imbalance problem" [112, 114], we created a dataset composed of real and synthetic license exception instances. In particular, given a set of software licenses text L, and a set of classes[1] representing the possible exceptions in licenses, we define a data instance for the classification process as a couple $d_i = \langle l_i, e_j \rangle$ where l_i represents a license text and e_j the specific license exception declared in l_i. We consider 33 license exception types, including the ones in the SPDX index [143], the ones in Table 4.1, and two variants (one of the *FLTK* and one of the *Nokia QT GPL* exception).

As the classifiers should be able to distinguish when a license text l_i does not contain an exception, we included a negative class *Not-an-exception* to describe the case in which a data instance d_i is not representative of any exception (*e.g.,* the canonical text of GPL does not include a license exception). Therefore, our classifiers consider 34 possible classes to which a license text l_i can be assigned - 33 for the exceptions plus the *Not-an-exception* class.

Also, note that a license text l_i is assigned to only one e_j, and the motivation for this is that in our first study we did not find licenses with more than one exception. Therefore, the classification process we are conducting is single-label.

The procedure we followed to build the dataset for evaluating the proposed approach is the following:

1. For each e_j (*i.e.,* for each possible class) we build an empty bucket $B[e_j]$; each bucket contains the corresponding d_i instances from the real examples and the generated synthetic instances.

2. We assign the empirically found license exceptions (*i.e.,* $d_i = \langle l_i, e_j \rangle, e_j \neq$ *Not-an-exception*) to the corresponding bucket $B[e_j]$.

[1] "Class" refers to the target of the classification process (exception type).

3. Given a target sample size s to achieve within all the buckets, we fill each bucket $B[e_j]$ with synthetic instances until $|B[e_j]| = s$. A synthetic example for $B[e_j]$ is generated by randomly picking a canonical license (as indicated by the Open Source Initiative [144]) and appending the exception text of e_j after the license. This decision is based on the fact that in our previous study we did not find cases with an exception preceding a license; additionally, an exception applies to a license so it is reasonable to expect the license attribution prior to issuing an exception to the license.

4. For the examples in the negative class (*i.e.,* $e_j = \textit{Not-an-exception}$), we fill the corresponding bucket by randomly picking canonical licenses, without adding the exception at the end. We do not aim at identifying the type of a license, since existing license identification tools can deal with this task.

5. To ensure diversity of the licenses' text (including the exception text), each element in $B[e_j], \forall j \in [1, 34]$, is perturbed/mutated by randomly injecting typographical errors. We set a threshold of 1% of the words to be mutated, while also ensuring that at least one word was mutated. Prior work shows that scholarly writing texts can achieve an error rate of 0.2% [131], or 1.1% per word [92]. Therefore, to be conservative, we picked 1% as for the mutation rate in the texts (a developer is also less likely to review a header comment with the same detail as a publish work). These text mutations simulate slight changes that may occur in real data (*e.g.,* different copyright years or different copyright owner). Additionally, typographical errors are reasonable, as in the case of a license added by the National Institute of Standards and Technology in which there is a typo having "Untied Stated" instead of "United States" [21]. Finally, these mutations also allowed us to verify that our dataset did not have duplicated samples in the training and validation sets by computing SHA1 checksums of the files in both the training and test data.

6. Finally we split the data from each bucket into training and validation sets, by assigning 70% for training and 30% for validation.

Table 4.2: The distribution of the unique license exception instances in our "real data" for the evaluation of RQ_3. Asterisks signify non-SPDX exceptions.

Exception	**Unique Instances**	**Exception**	**Unique Instances**
Autoconf 2.0	22	Libtool	1
OpenSSL	11	MIF	1
Bison 2.2	2	Nokia Qt GPL*	1
Nokia QT LGPL	2	RACC*	1
Classpath 2.0	1	Rails Usage*	1
dh-make*	1	TexInfo*	1
GUILE*	1	WxWindows	1

In addition to the synthetic dataset, we built a second dataset to be used as test set. This test set is composed only of real examples of canonical licenses and exceptions that we found in our preliminary study in the analyzed GitHub projects. Our goal with this dataset is to measure the generalization error [83] of the classifiers and of the baseline on "real data". To construct this dataset, we identified the unique instances from the results of our first study by computing the SHA1 (Secure Hash Algorithm 1) checksum. We only included one representative example of each unique instance. Table 4.2 shows the breakdown of the dataset consisting of real exceptions we used for evaluating the classifiers and baseline accuracy. For the canonical licenses, we added an instance of each canonical license to avoid any bias induced by choosing a subset and represent the *Not-an-exception* class.

4.2.3 Building the Classifier

To build a classifier, we first extract terms from the licensing statements of the files under analysis. We perform a preprocessing in which we (i) remove English stop words, and (ii) weight the terms using the *tf-idf* weighting scheme [88].

Then, we use the data from our training set to build a machine learning classifier, using the license words as features (weighted by their *tf-idf*) and as a dependent variable the (manually labeled) kind of exception contained in the license.

We consider four machine learning classifiers that have been widely used for text categorization [133, 84]: decision trees (DT), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM). To build the classifiers, we relied on the Weka [109] data mining library. The machine learners aim to classify the dataset into 34 classes: six new exceptions from our preliminary study (marked with an asterisk in Table 4.2), 25 from SPDX's index [143], one *FLTK Exception* variant, *Nokia QT GPL Exception* variant, and the *Not-an-exception* class.

4.2.4 Analysis Method

To answer $\mathbf{RQ}_2$, we compared the accuracy of the classifiers in terms of the F-1 score [138], which is the harmonic mean of precision and recall, and the Receiver Operating Characteristic (ROC) area [101]; both metrics are widely used in the machine learning community to evaluate classifiers with an unified metric [138, 101]. The accuracy of the classifiers was measured with the synthetic dataset described in Section 4.2.2 and with a dataset composed only of the real examples we found in our preliminary study. The evaluation on both datasets has the objective to verify whether our results are an artifact of the synthetic dataset. To measure the sensitivity of the classifiers to the sample size, we trained/tested the classifiers with samples sizes of 100, 200, 300, 400, 500, 1K, 2K, 3K, 4K, 5K, and 10K for each class (*e.g.,* $|B[e_j]| = 10,000, \forall j \in [1, 34]$). The results for $\mathbf{RQ}_2$ are reported in Section 4.2.5.

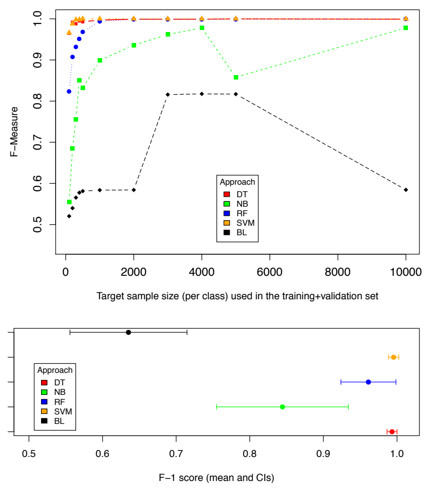
Concerning $\mathbf{RQ}_3$, since there is no existing approach for identifying license exceptions, we constructed a baseline (BL) to compare against the machine learning classifiers. The baseline approach attempts to search for the license exception text of each license exception. Since we already demonstrated in our preliminary study that a keyword-matching approach is prone to low precision (see Section 4.1—40.9% of precision), we considered a more conservative baseline that matches the entire exception text. As for the target text to match, for each exception, we extracted a canonical example from the real data, and from the SPDX list when no instance was available in the real data. Subsequently, the approach matches text of the license exception to ensure the entire text is contained in the test file. If the match is confirmed, the text under test is tagged as containing the exception. As for **RQ$_2$**, the comparison is done in terms on the F1-score achieved by the baseline and the classifiers, and by performing sensitivity analysis with different sample sizes. The results for **RQ$_3$** are reported in Section 4.2.6.

For both **RQs**, we use statistical tests to measure the significance of the achieved results. In particular, we use the Wilcoxon signed-rank test [134] (with $\alpha = 0.05$) in order to statistically compare each approach with the baseline across the different sample sizes. We use the (paired) Wilcoxon test as the comparisons (*e.g.,* SVM *vs.* BL) are performed between paired samples. Since we perform multiple pairwise comparisons, we adjust p-values using the Holm's correction procedure [111].

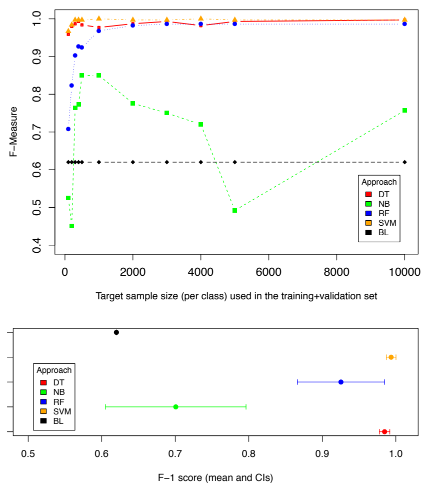
In addition, we estimate the magnitude of the observed differences by using the Cliff's Delta (d), a non-parametric effect size measure for ordinal data [108]. Cliff's d is considered negligible for $d < 0.148$ (positive as well as negative values), small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$ [108]. Finally, to visually corroborate the significance of the differences and the overlapping between the accuracies achieved by each approach, we computed the confidence intervals of the accuracies with 95% of confidence.

4.2.5 Results for RQ$_2$: What classifier provides the best accuracy for license exception identification relying on ML?

Synthetic data. Figure 4.1a shows the F-1 scores achieved across different sample sizes for each classifier, and the confidence intervals around the means of the F-1 scores, on the validation set from the synthetic data. Table 4.3 reports the results of the Wilcoxon tests (adjusted p-values and Cliff's d) for each pairwise comparison. In general, there are statistically significant differences between NB and the other classifiers; this is confirmed by the F-1 curves, the ROC areas, the Wilcoxon tests, and the confidence intervals. Concerning the other three classifiers (*i.e.,* RF, SVM, and DT), the results suggest that SVM and DT, in general, outperform RF when training with samples sizes per class less than



(a) Synthetic data (Validation set)



(b) Real data from GitHub projects (Test set)

Figure 4.1: F-1 scores and confidence intervals achieved across different sample sizes when testing the analyzed approaches with a) synthetic data (*i.e.*, validation set), and b) license exceptions extracted from GitHub projects (*i.e.*, test set).

1K. DT and SVM are close in terms of performance for all the sample sizes and their confidence intervals completely overlap each other and the difference is marginally significant (*p*-value=0.049). In the dataset with 1K samples per class, the difference between SVM, decision trees, and random forests become minimal. The ROC area is always $> 90\%$ for all the classifiers except for NB when the dataset has 100 instances per class. Both DT and SVM exhibit a high precision and recall for all sample sizes. For a sample size of 200 DT outperforms SVM, and they are basically equivalent for a sample size of 5K. While random forests initially ranked third, it begins outperforming decision trees at sample size of 2K, excluding a performance drop in performance for a sample size of 5K. NB always had the lowest precision and recall.

SVM is able to correctly classify 96.28% of the validation set in the worst case, and classifies 99.99% of the validation data correctly in the best case. The lowest precision achieved by SVM is 0.974 and the highest is 1 (a value of 1 is achieved when the incorrect classification was between 0.01% and 0.04%). We observe this drop in terms of magnitude (from 0.20% to 0.02%) for a sample size of 500. The best results for SVM are for a sample size of 2K when it only misclassifies 2 license exceptions. In fact, we observe the number of incorrectly classified exceptions decreases from sample size 100 until 1K when it rises before dropping again at a sample size 2K. After sample size 2K, the percentage of incorrect license exceptions rises from 0.01% to 0.04% of the test set.

We observe that decision trees, random forests, and SVM have a relatively similar accuracy in classification starting at a sample size of 2K. While SVM outperforms both random forest and decision trees. Decision trees incorrectly classify between 0.03% and 0.08% of the test set, while random forests incorrectly classify between 0.05% and 0.06% of the testing data, while SVM incorrectly classifies between 0.01% and 0.04% of the test set.

Real data. The results of the supervised learners on the real dataset (see Figure 4.1b and Table 4.3) demonstrate a similar behavior to their results on the validation dataset. In general, SVM outperforms the other supervised learners on the real data (although DTs marginally outperforms SVM for sample sizes of 100 and 200). SVM achieves an F-1 score of 0.997 or 1 for sample sizes 300 and larger (the precision is also 1 for each of these evaluations). In smaller sample sizes, we observe that DTs outperform RF, but this behavior switches above training sample size of 2K (inclusive of 2K). The ROC area is always $>$ 90% for all the classifiers except NB when the dataset has 100, 300, 4K, and 5K instances per class.

Summary for RQ$_2$. *The results demonstrate that SVM, DT, and RF outperform NB with statistical significance in both validation and testing set. SVM also attains a higher precision and recall than DT and RF. In terms of the best accuracy, in the validation set, SVM is the first to achieve a ROC area of 99.99% at sample size of 2K, and a F-1 measure of 1 for datasets of at least 500 instances per class. In the real test set, SVM always exhibits a F-1 scores greater than 0.95, and a ROC area greater than 97%.*

4.2.6 Results for RQ$_3$: Can our machine learning-based approach beat a baseline approach matching the license exception text?

Synthetic data. The baseline approach exhibits high precision for all sample sizes in the dataset, ranging between 0.958 and 0.976. This result is expected since the approach relies on matching the text of the exception. Thus, it should find the correct match when the exception is directly copy-pasted. The negative class suffers from a high number of false positives, which harms the overall precision of the baseline approach (it wrongfully marks the license exceptions as licenses). The recall suffers since this type of baseline does not account for changes like modifications to a copyright year, copyright holder, or typographical errors in the exception. We observe a recall as low as 0.361 for a sample size of 100, and it achieves a maximum value of 0.677 for sample size 4K. However, we observe a decrease in precision at sample size 5K and 10K; the latter case drops to 0.418.

SVM outperforms the baseline in terms of both precision and recall. The precision of SVM and the baseline approach at sample size 100 is very close, only differing by 0.016, while the largest difference in precision is 0.0529 at sample size 4K. However, the recall is much lower for the baseline approach, never able to outperform any of the classifiers in terms of precision and recall for any of the sample sizes. SVM has more than twice the recall as compared to the baseline for samples sizes of under 2K (inclusive of 2K) and sample size 10K. In the best case, SVM still outperforms the baseline in terms of recall with a difference is approximately 0.3. In terms, of statistical tests, SVM has a significantly higher classification performance when compared to the baseline (p-value $<$ 0.05 with a large—0.909—effect size).

Real data. In addition to the synthetic comparison, we compare the baseline and the supervised learners on the real dataset (see Figure 4.1b and Table 4.3). The real dataset consists of the unique license exceptions identified in the preliminary as well as an example of each canonical license (canonical licenses are for the negative class). The baseline (textual-based matching) achieves a precision and recall of 0.73 and 0.54, respectively. We observe that the baseline is able to outperform NB in terms of recall and F-1 score for the NB classifier trained on sample sizes of 100 and 200, and the baseline is able to outperform NB in terms of precision for sample size of 5,000, which is the only case of the baseline achieving a better precision than a supervised learner.

Figure 4.1b depicts the F-1 score and confidence intervals for supervised learners and baseline when classifying the real data. We observe that the DT and SVM have similar confidence intervals that are tight around the F-1 score. Only NB and the baseline do not exhibit a significant difference in terms of performance. These results are confirmed by the confidence intervals shown in Figure 4.1b. These results indicate that the supervised learners trained on synthetic data can achieve a high precision and recall with respect to classifying license exceptions and identifying the absence of an exception (*i.e.,* the negative class). Therefore, a supervised learner integrated after the initial license identification stage would enable a license compliance engine to determine if the particular license includes some additional exception to its terms.

Summary for RQ$_3$. *The results show that supervised learners trained on synthetic data were able to outperform a baseline approach when classifying synthetic and real data. Specifically, supervised learners are able to handle variations, which occur in practice. Furthermore, the supervised learners outperform the baseline approach on real data.*

4.2.7 Threats to Validity

Construct validity threats can be mainly due to bias when sampling our datasets. We balanced the classes of our dataset and generated synthetic license and license exception pairs randomly. Additionally, we performed sensitivity analysis by varying sample sizes between 100 instances per class to 10K instances per class. *Internal validity* threats can occur in the creation of the training and test sets. When evaluating the performance of the classifiers, we considered 70% of the dataset for training and 30% for validation, which is an accepted practice for evaluating supervised learners. Additionally, we had a separate testing dataset with real data. For what concerns *external validity* threats, it is possible

Table 4.3: Results of the Wilcoxon test (adjusted p-values and Cliff's d effect size) for the pairwise comparisons between the classifiers (DT, NB, RF, SVM) and the baseline (BL), when using the synthetic data (*i.e.,* validation set), first table, and when using real license exceptions found in GitHub project (*i.e.,* test set), second table.

Synthetic Data					
Comparison	p**-value**	**Cliff's** d	**Comparison**	p**-value**	**Cliff's** d
DT vs. NB	0.019	0.884	NB vs. RF	0.010	-0.652
DT vs. RF	0.049	0.214	NB vs. SVM	0.019	-0.966
DT vs. SVM	0.049	-0.454	NB vs. BL	0.010	0.686
DT vs. BL	0.009	0.909	RF vs. SVM	0.019	-0.561
SVM vs. BL	0.010	0.909	RF vs. BL	0.010	0.909

Real Data					
Comparison	p**-value**	**Cliff's** d	**Comparison**	p**-value**	**Cliff's** d
DT vs. NB	0.010	0.909	NB vs. RF	0.010	-0.835
DT vs. RF	0.022	0.595	NB vs. SVM	0.010	-1.000
DT vs. SVM	0.022	-0.669	NB vs. BL	0.083	0.455
DT vs. BL	0.022	0.909	RF vs. SVM	0.022	-0.835
SVM vs. BL	0.021	0.909	RF vs. BL	0.022	0.909

that the performance of the classifiers on the synthetic dataset does not generalize. Such a dataset was designed to replicate user errors and aimed to ameliorate the problem of limited real data.

The comparison with the real data suggests that the perturbation rate of the dataset may inflate the differences between to licenses with exceptions. It is possible that results might vary using different datasets (*e.g.,* from other projects).

4.2.8 Discussion

This study addressed a novel problem of detecting license exceptions. Our results indicate the effectiveness of supervised learners in identifying (when present) different types of license exceptions reported in licensing statements. In general, SVM seems to be the most applicable supervised learner, since it outperformed the other algorithms. The results from the evaluation on the real data mimics our results on the synthetic data, which contains perturbations. The findings suggest that supervised learners are able to learn relationships between the terms of a license exception, while allowing for variations.

Results also indicate that keyword-based identification of license exceptions generate a large number of false positives (59.1%), which is not acceptable for an automated tool aimed at supporting software developers, especially in large software systems. Similarly, the recall of a baseline approach by matching the text of the license exception results in a low recall so it is likely to miss license exceptions. It is important to note that no other approaches exist and the baseline was designed to represent how a developer could identify license exceptions (*i.e.,* by pattern matching) without having to devise a sophisticated approach.

4.2.9 Related Work

Our work is related to prior approaches on license identification and previous empirical studies on software licenses. Our work is novel in that it is the first one to investigate the presence of license exceptions in software and to classify them.

License Identification. Several approaches exist to identify the license type and version of source code and jars, but these approaches do not address license exceptions [107, 146, 99, 102]. The FOSSology project [107] first utilized machine learning to classify licenses in order to solve the challenge of license identification. ASLA was also proposed by Tuunanen *et al.* [146] with a high accuracy of 89%. *Ninka*, the state-of-the-art approach proposed by German *et al.* [106] utilized sentence matching and was empirically shown to have a precision of 95%. Di Penta *et al.* [99] sought to identify the licensing of jars and proposed an approach employing code search (Google Code, which is no longer available). Lastly, German *et al.* [102] investigated the impact of propriety licensing, when used with free/open source licenses, on the ability to accurately identify the free/open source license by analyzing 523,930 archives.

It is important to note that these previous approaches focused on the identification of licenses and not on the identification of license exceptions. Our work aims to complement these approaches in order to evaluate whether the identified license contains an exception or if it is a canonical license. This analysis is vital for license compliance, since it refines the terms of modification or redistribution of the system.

Empirical Studies. German *et al.* [105] identified exceptions as a viable method for licensors to modify a certain license. In that work, the authors proposed a model, based on their investigation of 124 free/open source packages, to illustrate the applicability of particular licenses. However, the work does not empirically investigate the existence of license exceptions. German *et al.* [103] studied license inconsistencies in Fedora-12 distribution. The authors investigated potential inconsistencies across dependencies and between packages in the distribution. Importantly, this work demonstrated the importance of studying license exceptions since the results showed that license exceptions are an important factor for validating potential license inconsistencies.

Additionally, Manabe *et al.* [119] investigated license changes in FreeBSD, OpenBSD, Eclipse, and ArgoUML and found the change patterns were project specific. Vendome *et al.* [152] investigated 16,221 free/open source Java systems to understand the license usage and changes in licensing. Additionally, the work investigated the reason for potential usage and changes by analyzing commits and issues trackers. Vendome *et al.* [153] also performed a survey involving software developers to investigate *when* developers pick a particular license or changes the license(s) and to understand the underlying reasons *why* developers choose or change licensing of their system.

Other empirical studies focused on license inconsistencies in code clones between Linux and either OpenBSD or FreeBSD [104] and inconsistencies between the licensing of code clones in Debian 7.5, suggesting potential violations [154].

4.3 Discussion

We studied—for the first time, to the best of our knowledge—the presence of license exceptions in 51,754 free/open source systems from six languages. We found 14 different license exception types across 298 files in five of the six languages. While we observed that certain license exceptions are more prevalent in projects written in specific languages, we also found that the *Autoconf Exception* was within the top-3 most prevalent license exceptions for all of the languages (being Autoconf, a cross-language tool). Additionally, we also observed the frequent coexistence of the *RACC Exception* and *Rails Exception* due their presence in the same library. These license exceptions directly impact the way in which software can be reused and are critical for understanding license compliance. Specifically, these exceptions ameliorate inconsistencies under particular circumstances. We have sent the new license exceptions for consideration to the SPDX team.

After that, we evaluated the applicability and effectiveness of supervised learners for the identification of license exceptions. The results indicate that machine learning classifiers—and specifically SVM and Random Forests—are able to achieve high precision and recall when identifying the type of license exception as well as determining the lack of a license exception on real and synthetic data. A license exception classifier can be integrated into a license compliance engine after the initial license identification. In our future work, we aim to create a solution that integrates an SVM-based classifier for classifying exceptions into a license identification tool.

4.4 Bibliographical Notes

The paper supporting the content described in this Chapter was performed in collaboration with other members of the SEMERU group at William and Mary and researchers from the Universidad de los Andes, University of Sannio, University of Lugano, and the University of Victoria:

Vendome, C. Linares-Vásquez, M., Bavota, G., Di Penta, M., Germán, D., and Poshyvanyk, D., "Machine Learning-Based Detection of Open Source License Exceptions", in Proceedings of 39th IEEE/ACM International Conference on Software Engineering (ICSE'17), Buenos Aires, Argentina, May 20-28, 2017, pp. 118-129.

Chapter 5

Automatically Determining License Compatibility Across Dependencies

From our prior work outlined in Chapters 2 and 3, we observe a need for a license compliance tool to assist developers with ensuring their system complies with all attributed licenses. We observed that developers struggle understanding the terms of licenses and how licenses can be used together [153, 150]. Further complicating issues, free/open source software licenses and their compatibility are not always interpreted consistently and so communities also create guidelines with respect to licensing [150]. Because of this difficulty, any incompatibilities need to be presented in such a way that is easy for a developer to understand the problem. Additionally, a prior work also demonstrated the prevalence of license incompatibilities of projects and their dependencies for Android projects on F-droid [125]. To assist developer with this process, we built an infrastructure to a evaluate the compatibility between licenses and automatically identify potential license violations between a system and its entire dependency tree. Since software needs to comply with all of the licenses of the code it reuses and redistributes, it is important to consider all of the dependencies. For example, if System$_A$ depends on System$_B$ and System$_B$ depends on System$_C$, then System$_A$ must comply with the licensing terms of both System$_B$ and System$_C$. Similarly, if System$_C$ contains some license incompatibility, then it would propagate to System$_B$ and subsequently System$_A$.

The infrastructure first performs the license identification step to extract the licensing of all of the files utilizing ninka [106] and the proposed approach from Chapter 4 to identify the licenses and exceptions. Subsequently, we identify the dependencies by parsing the build files (e.g., pom.xml). We then identify the licensing of the dependencies in order to evaluate the compatibility between the system and its dependencies. We utilize a pair-wise matrix, which evaluates if two given licenses are compatible, to perform the license compatibility analysis. It is important to note that the matrix is not symmetric since the compatibilities between licenses are directed. Finally, the infrastructure returns a list of potential incompatibility warnings or it returns that there were no incompatibilities.

In this Chapter, we present a study utilizing the infrastructure to build dependency trees and to perform the license compatibility analysis across those trees. We aim to answer the following research questions:

- *How prevalent are potential license incompatibilities between a project and its dependency along the entire dependency tree?*
- *Which license combinations are responsible for these potential license incompatibilities?*

In order to answer these research questions, we analyzed over 98 million dependency relationships belonging to more than 3.6 million projects. We recursively extracted the dependencies of these projects to build dependencies as well as extracting the unique set of all dependencies of a certain system throughout its dependency tree. With this information, we utilize the system's licensing or the dependency's licensing to perform the compatibility analysis between the pairs of licenses. When present, we extract both the dependency and the licenses that introduces the license incompatibility. It is important to note that this work differs from [125] in several ways: i) the prior work considered a 14 licenses pairs for their compatibility analysis, while this work considers the relationships between 54 different licenses; ii) this work considers projects in 6 different languages; iii) this work considers the compatibility between all of the dependencies along the dependency chain.

It is important to note that we refer to these as potential license incompatibility warnings and an in-depth analysis is needed to further complement this work to better evaluate the system's design in order to confirm the warnings. We observe that incompatibilities between licenses is quite prevalent in our dataset. We found that 686 projects in our dataset had a warning of an incompatible license with their dependencies. The mean number of warnings of license incompatibilities per system is 15 and the median number of warnings of license incompatibilities per system is 3.

5.1 Evaluating Dependency Trees for License Compatibility

To facilitate a large-scale analysis of multiple ecosystems, we considered the dataset made available by libraries.io [115], which contains 98,269,642 dependency relationships spanning 3,686,361 projects from 36 package managers and 160 programming languages. In addition to the dependency relationships, it contains project metadata, which includes the licensing information for 2,215,454 projects. Since the number of dependencies exceeds the number of projects containing metadata, any missing projects are marked to have an unknown license. From this dataset, we sampled 6 of the 15 most popular programming languages according to GitHub's 2017 analysis [37]. For each project, we recursively build a dependency tree by traversing all of the project-level dependency relationships. Specifically, we consider System_A and extract all of the dependencies. Then, for each dependency of System_A, we extract their dependencies. We repeat this process iteratively until there are no more dependency relationships or we re-visit an already identified dependency (i.e., if System_B and System_C both depend on System_D, they would both add System_D as a dependency node, but only one instance would fully recurse System_D). The latter condition serves two purposes: i) it serves as a condition to avoid cycles that might generate an infinite length tree, and ii) it avoids redundant analysis of the same dependencies. In the presented example, System$_A$ is compared against all of System$_D$'s dependencies when it analyzes System$_B$'s dependencies; thus, it is not necessary to perform this analysis a second time when analyzing System$_C$'s dependencies.

Figure 5.1 illustrates what a potential dependency tree might look. To build this tree, we first create the root, which represents the system that is being analyzed. We then extract its dependencies and create a node for each dependency, linking the dependency to the dependent system with an edge (the arrow indicating a node is a child). Those added dependency nodes are then analyzed in the same way to extract and link their dependencies. As previously stated, this iterative process continues until either all of the dependencies are linked or a dependency that was already linked is extracted. In Figure 5.1, the root of the tree is the system under analysis (System$_A$) and it contains edges to its dependencies (System$_B$ and System$_C$). Since System$_D$ is in the set of dependencies for both System$_B$ and System$_C$, they both have an edge to a node representing System$_D$. However, only one node representing System$_D$ contains edges to its dependencies (System$_E$ and System$_F$).

With these dependency tree, we evaluate the compatibility of the system and its dependencies by performing pair-wise license compatibility comparisons. To accomplish this pair-wise analysis, our infrastructure built upon an existing licensing compatibility tool [82] to perform the compatibility analysis between a given system and each of its dependencies in the system's dependency tree. For example, if we consider Figure 5.1, the infrastructure would check System$_A$'s compatibility with System$_B$ then System$_C$, individually. It would then evaluate the compatibility with System$_B$ and Systems$_C$'s dependencies. As they both depend on System$_D$, we only need to check compatibility with System$_D$ once. Then, the licensing of System$_A$ is checked against the licensing of System$_D$'s dependencies for compatibility. We identify the conflicting dependencies and extract the conflicting licensing pair (e.g., *GPL-2.0* and *Apache-2.0*). Similarly, we can identify the dependencies

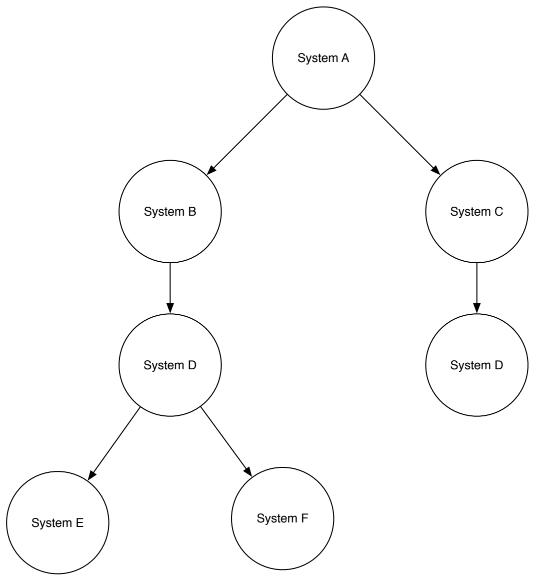


Figure 5.1: Illustrative Example of a Dependency Tree.

that have licenses conflicting with each other by performing this same analysis between the dependencies. With these pieces of information, we can provide the developer with warnings of the dependencies that seem to conflict with the system that is being developed as well as warnings of dependencies that seem to conflict with each other. The latter case can be useful to a developer when trying to fix the incompatibility, since this conflict may inhibit migrating licenses as a solution.

5.2 License Inconsistencies Results

When analyzing a project's license against its dependencies along the whole dependency tree, we considered projects with at least one warning of a license incompatibility and we excluded projects that included non-essential frameworks (i.e., they would not be distributed, like unit testing) and runtime environments in their dependency tree as these both would generate many false positives. This resulted in a total of 686 projects (7.03%) having license incompatibility warnings with a dependency. For these projects, we also investigated how many incompatibilities were present. The number of license incompatibilities ranged from 1 warning (as expected since we are considering the subset with at least one incompatibility) to a maximum of 35 incompatibility warnings. The mean and median number of incompatibility warnings are 17 and 3, respectively. Thus, we observe that it is common for projects to have multiple potential license incompatibilities.

Table 5.1 shows the number of projects with license incompatibility warnings for the 6 most popular programming languages according to GitHub's 2018 analysis [37]. Overall, we observe that 7.03% of these projects contain a warning for a potential license incompatibility. By language, we observe a range from 0.20% (C#) to 24.08% (Swift) of projects with license incompatibility warnings. Interestingly, we observe that Java, C++, C, and Objective-C have similar rates of potential license incompatibilities, while C# and Swift differ greatly on the extremes. In the case of C#, we only observed 4 projects (0.20%) with warnings of potential license incompatibilities, while almost a quarter of Swift projects had license incompatibility warnings.

We observed a total of 13,787 warning for potential license incompatibilities between a project's license and the licensing of their dependencies. Table 5.2 shows the top 10 pairs of licenses with the most license incompatibility warnings. We observe that the *MIT* license and *GPL-3.0* license was the most prevalent potential license incompatibility with 6,077 occurrences (44.08%). In fact, the top-4 license incompatibility pairs account for 85.43% of the license incompatibility warnings. However, the *Apache-2.0* and *BSD-3-Clause* were also among the top 10 with a dependency licensed under a version of the *GPL*. Interestingly, the 4th most frequent pair resulting in a license compatibility was the pair in the top 10 that was not between a permissive license and the *GPL*, but it was a permissive license (*MIT*) and the *AGPL*.

5.3 Threats to Validity

Construct Validity: To reduce the impact of this threat to validity on evaluating license compatibility, we built our approach upon an existing license compatibility tool. While

Table 5.1: Frequencies of license incompatibility warnings by language compared against the total number of projects for each language

Language	**#Warnings**	**Total Projects**	**%Warnings**
Java	256	5,283	4.85%
C++	12	210	5.71%
C#	4	1,976	0.20%
C	20	306	6.54%
Swift	361	1,499	24.08 %
Objective-C	33	485	6.80%
Total	686	9,759	7.03%

Table 5.2: Top 10 license violation pairs between a project and dependencies in its dependency chain

Project License	**Dependency License**	**Occurrences**
MIT	GPL-3.0	6,077
MIT	GPL-2.0	3,600
MIT	GPL-2.0+	1,500
MIT	AGPL-3.0	601
Apache-2.0	GPL-3.0	587
Apache-2.0	GPL-2.0	261
BSD-3-Clause	GPL-3.0	209
Apache-2.0	GPL-3.0+	144
BSD-3-Clause	GPL-2.0	120
Apache-2.0	GPL-2.0+	101

the tool was developed by analyzing the compatibility of certain licenses, it has limitations in the number of license that it can evaluate for compatibility. Additionally, a subset of projects have missing licensing information. Thus, the analysis might miss certain incompatibilities. However, it includes the most popular free/open source software licenses. Additionally, the extracted dependencies were identified through build files. If these files are outdated and not maintained, it is possible some dependencies might have been removed or might have been added that are not reflected in the build files. However, these results represent a snapshot at a point in the system's development; thus, it demonstrates the potential incompatibilities that were there at that point in development. Additionally, we present these results as warnings for potential license incompatibilities, since certain architectural design decisions may address the potential license incompatibility. Similarly, it serves as a warning to a developer since the violation of the license occurs when the developer distributes the software.

Internal Validity: To reduce this threat, we evaluate the number of warning of incompatibilities from a set of unique dependencies. This step prevents inflating the number of warnings from one or a few popular libraries that might appear multiple times in the dependency tree. Additionally, we present the results for the prevalence of projects with warnings of potential license incompatibilities in terms of a binary option (warning or no warning) to avoid misrepresenting the number of warnings when a project may have multiple warnings for different dependencies along its dependency tree.

External Validity: We analyzed a large dataset that is cross-language and spans many popular package managers, but other free/open source projects from different communities may result in different observations. The diversity of our dataset in terms of languages and package mangers as well as the number of projects analyzed gives us confidence in our findings with respect to software that are libraries. However, the dataset predominantly is focused on libraries and as such these results may not generalize to all software projects. A broader analysis on other types of systems is required. Also, to generalize these finding further, it would also be important to conduct license compatibility analysis in closed source systems as these may also go through a more stringent auditing process, especially for organizations with strong legal departments.

5.4 Discussion

We presented an infrastructure to evaluate license compatibility in order to warn developers of a potential license incompatibility between their system and a dependency along the dependency tree. We utilized the state of the art in license identification [106] and our prior work detecting license exceptions [151] to identify the licensing of a system.

The infrastructure utilizes build files to extract the dependencies and subsequently build the dependency trees. Then, we incorporated a license compatibility tool [82] to evaluate the compatibility between the licenses within a project and the licenses across the dependencies in the dependency chain.

The license compatibility infrastructure was evaluated on a diverse dataset [115] and it uncovered 686 projects with license incompatibility warnings among projects written in 6 of the 15 most popular programming languages. We observed that the majority of these license incompatibility warnings were between projects licensed under the *MIT* license and reusing libraries licensed under a version of the *GPL* or the *AGPL*. This work demonstrates that developers could benefit from such a tool to ensure that their software complies with all of the licenses across their dependencies. It also demonstrates that a developer should be careful when reusing software because they may use a project that is seemingly compatible that has incompatibilities along its dependency chain. Future tools to support developers in identifying replacement libraries could be very useful to help developers remedy these license incompatibilities. Similarly, it would benefit developers to extend this analysis to evaluate license compatibility with binaries.

5.5 Bibliographics Notes

The work in this Chapter was performed in collaboration with other members of the SEMERU group at William and Mary and researchers from the Universidad de los Andes, University of Sannio, University of Lugano, and the University of Victoria. The study will be submitted to a future conference.

Chapter 6

Conclusion

The proposed dissertation makes several research contributions in four main areas: empirical studies, categorizing licensing bugs, license exception identification, and a novel practical approach for supporting developers with respect to software license compatibility.

- **Empirical Studies:** The empirical studies presented in Chapter 2 analyzed the difficulty and lack of knowledge that developers have when it comes to free/open source licensing. The results first aim to understand diffusion of licenses and then how these license migrate in evolving systems. In addition to compliance and compatibility constraints, developers are also influenced by external factors such as personal bias and community participation.These results demonstrate that automated support is important for developers (as we propose) and that researchers should aim to help developers with achieving proper licensing while considering external factors (*e.g.,* recommending dependencies that also meet licensing preferences or project goals).

- **Licensing Bugs:** The catalog presented in Chapter 3 demonstrates the issues and difficulties that developers face with software licensing. These issues are not isolated to reusing software, but it includes non-source artifacts and interpretations of the licenses and the implications of those interpretations. Additionally, it outlines how communities aim to avoid licensing issues by creating guidelines. The work suggests that developers would benefit from more education in terms of interpreting and understanding software licenses as well as the licenses that can be used together. The work also illustrates the need for better tool support to assist developers with uncovering license incompatibilities.

- **License Exception Identification:** This work proposed a machine learning-based approach to identify license exceptions, which can complement the existing licensing identification tools. These license exceptions are important for truly understanding compatibility, since they can change the permissions or restrictions of the applied license. The approach itself can be adapted to be utilized after existing techniques, as it does not rely upon a particular tool, in order to more accurately understand the licensing of a system.

- **License Compatibility:** We build an infrastructure to evaluate license compatibility between a software system and its dependencies along its dependency chain. We utilize the infrastructure to conduct a study on free/open source systems to understand the prevalence of potential license incompatibilities across many languages and ecosystems. We were able to identify 686 free/open source projects that had warnings of potential license incompatibility with at least one of their dependencies. Among 6 of the 15 most popular languages, we see that overall 7.03% of projects have a license incompatibility warning (ranging from 0.20% to 24.08% when considering each language individually). Thus, this infrastructure would benefit developers across several popular programming languages to further investigate and remedy the license incompatibilities in their software.

This dissertation outlines a road map for future research relating to free/open source software licensing. The catalog of *licensing bugs* outlines several unaddressed areas that require further research to aid developers with software licensing. Additionally, the infrastructure built in this dissertation can be extended and adapted in several ways to help developers at different points in the development process.

Linking Binaries to Licensing. The infrastructure is able to evaluate the compatibility of licensing between components when that licensing is available. In the case of binaries, the licensing statements do not survive compilation and so additional work is necessary to link these binaries such that their licensing can be identified. Di Penta *et al.* demonstrated that a code search approach could be viable for identifying the licensing of jars [99]. Davies *et al.* proposed a technique to link binaries to their source code counterpart [97, 98]. The infrastructure can be extended to incorporate these approaches prior to the license compatibility analysis of the system's components.

Community License Enforcement. While the license compatibility matrix serves to evaluate pairs of licensing, communities can impose stricter interpretations or regulations on the software that they distribute. Chapter 3 demonstrates how communities, like Debian, evaluate *freeness* according to their own guidelines. The infrastructure could be adapted to support contributors of these communities so that they comply with these additional restrictions. Similarly, the infrastructure can be extended to assist developers with documenting the licensing of their software as some communities impose a standardized way of documenting the project's licensing.

Recommending Replacement Libraries. Currently, the infrastructure analyzes the system and identifies the incompatibilities in a post-mortem manner. However, the infrastructure can be extended to perform this compatibility analysis earlier before the developer has written a large amount of code to alert the developer of the potential license incompatibilities. Additionally, the feedback could be augmented by utilizing techniques from information retrieval and clone detection. With these techniques, it may be possible to recommend a library with similar functionality that is legally compatible. Thus, the extended infrastructure would aim to search a large corpus (e.g., mined projects from GitHub, Bitbucket, and GitLab) for similar code with a compatible license to the system that was analyzed.

Bibliography

[1] http://githut.info/.

[2] 123done issue 139. https://github.com/mozilla/123done/issues/139.

[3] Action mailer basics. http://guides.rubyonrails.org/action_mailer_basics.html.

[4] android-sensorium issue 11. https://github.com/fmetzger/android-sensorium/issues/11.

[5] Apache License, Version 2.0 (current) https://www.apache.org/licenses/. Last accessed: 2015/03/23.

[6] brackets issue 8337. http://github.com/adobe/brackets/issues/8337.

[7] Cuanto commit. https://github.com/ttop/cuanto/commit/a1e58f2c93de40ab304c494e05853957c549fd44.

[8] Czmq commit. https://github.com/zeromq/czmq/commit/eabe063c2588cde0af90e5ae951a2798b7c5f7e4.

[9] d3-armory issue 5. https://github.com/kovmarci86/d3-armory/issues/5.

[10] enigma2 commit. https://github.com/openatv/enigma2/commit/b4dfdf09842b3dcacb2a6215fc040f7ebbbb3c03.

[11] F-Droid. https://f-droid.org/. Last accessed: 2015/01/15.

[12] GitHub. https://github.com. Last accessed: 2015/01/15.

[13] GNU General Public License. http://www.gnu.org/licenses/gpl.html. Last accessed: 2015/01/15.

[14] Gradle.https://gradle.org/.

[15] gtksourcecompletion issue 1. https://github.com/chuchiperriman/gtksourcecompletion/issues/1.

[16] gubg commit. https://github.com/gfannes/gubg.deprecated/commit/4d291ef433f0596dbd09d5733b25d27b3a921cf4.

[17] Haml commit. https://github.com/haml/haml/commit/537497464612f1f5126a526e13e661698c86fd91.

[18] Intex issue 1. https://github.com/mtr/intex/issues/1.

[19] jackson-module-jsonschema issue 35. https://github.com/FasterXML/jackson-module-jsonSchema/issues/35.

[20] jquery-browserify issue 20. https://github.com/jmars/jquery-browserify/issues/20.

[21] Mediadescriptionimpl.java https://java.net/projects/jsip/sources/svn/content/trunk/src/gov/nist/javax/sdp/MediaDescriptionImpl.java?rev=2364.

[22] minixwall commit. https://github.com/booster23/minixwall/commit/342171fa9e9d769ce4aa48525142a569b34962f7.

[23] neunode issue 5. https://github.com/snakajima/neunode/issues/5.

[24] New license/exception request: Bsd-3-clause-nonuclear. http://bit.ly/1rFJBmT.

[25] Nimble commit. https://github.com/bradleybeddoes/nimble/commit/e1e273ff18730d2f8e0d7c2af1951970e676c8d1.

[26] Open Source Definition http://opensource.org/osd.

[27] Passenger issue 1482. http://github.com/phusion/passenger/issues/1482.

[28] patchelf issue 37. https://github.com/NixOS/patchelf/issues/37.

[29] PF: The OpenBSD Packet Filter. http://www.openbsd.org/faq/pf. Last accessed: 2015/01/15.

[30] Postgis commit. https://github.com/postgis/postgis/commit/4eb4127299382c971ea579c8596cc41cb1c089bc.

[31] pyelection issue 1. https://github.com/alex/pyelection/issues/1.

[32] python-hpilo issue 85. https://github.com/seveas/python-hpilo/issues/85.

[33] rcswitch-pi issue 17. https://github.com/r10r/rcswitch-pi/issues/17.

[34] Ros-comm commit. https://github.com/ros/ros_comm/commit/e451639226e9fe4eebc997962435cc454687567c.

[35] schevorecipe.db commit. https://github.com/Schevo/schevorecipe.db/commit/b73bef14adeb7c87c002a908384253c8f686c625.

[36] Software Package Data Exchange (SPDX). http://spdx.org. Llast accessed: 2015/01/15.

[37] State of the Octoverse in 2017 https://octoverse.github.com/. Last accessed: 2018/04/14.

[38] Steampp issue 1. https://github.com/seishun/SteamPP/issues/1.

[39] Sun microsystems - bsd license. http://bit.ly/1SDwnNL.

[40] svgeezy issue 20. https://github.com/benhowdle89/svgeezy/issues/20.

[41] tablib issue 114. https://github.com/kennethreitz/tablib/issues/114.

[42] Tardis commit. https://github.com/tardis-sn/tardis/commit/07b2a072d89d45c386d5f988f04435d76464750e.

[43] web-workshops issue 1. https://github.com/rosedu/web-workshops/issues/1.

[44] What is free software? https://www.gnu.org/philosophy/free-sw.html.

[45] wkhtmltopdf-qt-batch commit. https://github.com/alexkoltun/wkhtmltopdf-qt-batch/commit/9b142a07a7576afa15ba458e97935aac5921ef8d.

[46] 17 usc 105: Subject matter of copyright: United states government works http://uscode.house.gov/view.xhtml?hl=false&edition=prelim&req=granuleid\%3AUSC-prelim-title17-section105, 2017.

[47] Apache legal-discuss mail list. legal-discuss@apache.org, 2017.

[48] Apache legal message: ”greenmail” library purported to be asl may actually be lgpl; is a dependency in several asf projects https://issues.apache.org/jira/browse/LEGAL-206, 2017.

[49] Appframework issue 21. https://github.com/01org/appframework/issues/21, 2017.

[50] bootstrap-admin issue: Incompatible license https://github.com/aristath/bootstrap-admin/issues/12, 2017.

[51] Bugzilla installation list. https://www.bugzilla.org/installation-list/, 2017.

[52] Bugzilla issue: [ip] best practices for interfacing with libs that are not epl compatible https://bugs.eclipse.org/bugs/show_bug.cgi?id=246945, 2017.

[53] Bugzilla issue: mpage included non-free code https://bugzilla.redhat.com/show_bug.cgi?id=1295170, 2017.

[54] Bugzilla issue: Review request: limnoria - a modified version of supybot (an irc bot) with enhancements and bug fixes https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=1342747, 2017.

[55] Bugzilla. https://www.bugzilla.org/, 2017.

[56] Creative commons licenses. https://creativecommons.org/licenses/, 2017.

[57] Debian-legal mail list. https://wiki.debian.org/DebianLegal, 2017.

[58] Debian legal message: Re: Cddl, opensolaris, choice-of-venue and the star package ... https://lists.debian.org/debian-devel/2005/09/msg00440.html, 2017.

[59] Debian legal message: Re: Debian and cuba https://lists.debian.org/debian-legal/2005/03/msg00493.html, 2017.

[60] Debian legal message: Re: Debian-approved creative/content license? https://lists.debian.org/debian-legal/2007/03/msg00065.html, 2017.

[61] Debian legal message: Re: Debian-approved creative/content license? https://lists.debian.org/debian-legal/2007/03/msg00069.html, 2017.

[62] Debian legal message: Re: Help about texture inclueded in stellarium https://lists.debian.org/debian-legal/2004/07/msg00832.html, 2017.

[63] Debian legal message: Re: How long is it acceptable to leave *undistributable* files in the kernel package? https://lists.debian.org/debian-legal/2004/06/msg00381.html, 2017.

[64] Debian legal message: Re: How long is it acceptable to leave *undistributable* files in the kernel package? https://lists.debian.org/debian-legal/2004/06/msg00383.html, 2017.

[65] Debian legal message: Re: Ibm jikes license. https://lists.debian.org/debian-legal/1998/12/msg00107.html, 2017.

[66] Debian legal message: Re: Packaging the meego stack on debian - use the name ?. https://lists.debian.org/debian-legal/2011/01/msg00011.html, 2017.

[67] Debian legal message: Re: Trolltech gpl violation? https://lists.debian.org/debian-legal/2006/01/msg00011.html, 2017.

[68] Debian legal message: Re: Visualboy advance question. https://lists.debian.org/debian-legal/2004/06/msg00619.html, 2017.

[69] Debian legal message: Trolltech gpl violation? https://lists.debian.org/debian-legal/2006/01/msg00000.html, 2017.

[70] Fedora legal message: Re: [fedora-legal-list] status of icq-related apps in fedora. https://mid.mail-archive.com/legal@lists.fedoraproject.org/msg00185.html, 2017.

[71] Fedore-legal list. https://www.redhat.com/archives/fedora-legal-list/, 2017.

[72] Genome-legal list. https://mail.gnome.org/archives/legal-list/, 2017.

[73] Gnu free documentation license. https://www.gnu.org/licenses/fdl-1.3.en.html, 2017.

[74] Gpl compatibility. http://apache.org/licenses/GPL-compatibility.html, 2017.

[75] https://www.copyright.gov/help/faq/faq-general.html, 2017.

[76] An injunction against fortinet for gpl violations. https://lwn.net/Articles/132143/, 2017.

[77] Linux copying file: https://github.com/torvalds/linux/blob/master/COPYING, 2017.

[78] Mozilla trademark policy for distribution partners. https://www.mozilla.org/en-US/foundation/trademarks/distribution-policy/, 2017.

[79] Openstack legal-discuss. http://lists.openstack.org/cgi-bin/mailman/listinfo/legal-discuss, 2017.

[80] picotcp issue: Upgrade license to gplv2+ or gplv3+ https://github.com/tass-belgium/picotcp/issues/408, 2017.

[81] To distribute or not to distribute? why licensing bugs matter - online appendix https://sites.google.com/site/licensingbugsunveiled/, 2017.

[82] license-compatibility tool https://github.com/librariesio/license-compatibility, 2018.

[83] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from Data*. AMLBook Press, 2012.

[84] Charu C. Aggarwal and ChengXiang Zhai. *Mining Text Data*, chapter A Survey of Text Classification Algorithms, pages 163–222. Springer US, Boston, MA, 2012.

[85] Daniel A. Almeida, Gail C. Murphy, Greg Wilson, and Mike Hoye. Do software developers understand open source licenses? In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 1–11, May 2017.

[86] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA, 2006. ACM.

[87] Apache. Apache maven project. https://maven.apache.org/.

[88] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[89] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. The evolution of project inter-dependencies in a software ecosystem: The case of apache. In *2013 IEEE International Conference on Software Maintenance(ICSM)*, volume 00, pages 280–289, Sept. 2014.

[90] Gabriele Bavota, Alicja Ciemniewska, Ilknur Chulani, Antonio De Nigro, Massimiliano Di Penta, Davide Galletti, Roberto Galoppini, Thomas F. Gordon, Pawel Kedziora, Ilaria Lener, Francesco Torelli, Roberto Pratola, Juliusz Pukacki, Yacine Rebahi, and Sergio García Villalonga. 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 2014, Antwerp, Belgium, February 3-6, 2014*, pages 399–402, 2014.

[91] Amanda Brock. Project Harmony: Inbound transfer of rights in FOSS Projects. *Intl. Free and Open Source Software Law Review*, 2(2):139–150, 2010.

[92] François Chédru and Norman Geschwind. Writing disturbances in acute confusional states. *Neuropsychologia*, 10(3):343 – 353, 1972.

[93] Pamela Chestek. Who owns the project name? *International Free and Open Software Law Review*, 5(2), 2013.

[94] Jacob Cohen. A coefficient of agreement for nominal scales. *Educ Psychol Meas.*, 20:37–46, 1960.

[95] Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

[96] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pages 275–284. IEEE, 2014.

[97] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. Software bertillonage: Finding the provenance of an entity. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 183–192, New York, NY, USA, 2011. ACM.

[98] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. Software bertillonage. *Empirical Software Engineering*, 18(6):1195–1237, Dec 2013.

[99] Massimiliano Di Penta, Daniel M. Germán, and Giuliano Antoniol. Identifying licensing of jar archives using a code-search approach. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, pages 151–160, 2010.

[100] Massimiliano Di Penta, Daniel M. Germán, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 145–154, 2010.

[101] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[102] Daniel M. Germán and Massimiliano Di Penta. A method for open source license compliance of java applications. *IEEE Software*, 29(3):58–63, 2012.

[103] Daniel M. Germán, Massimiliano Di Penta, and Julius Davies. Understanding and auditing the licensing of open source software distributions. In *The 18th IEEE International Conference on Program Comprehension, ICPC 2010, Braga, Minho, Portugal, June 30-July 2, 2010*, pages 84–93, 2010.

[104] Daniel M. Germán, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16-17, 2009, Proceedings*, pages 81–90, 2009.

[105] Daniel M. Germán and Ahmed E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 188–198, 2009.

[106] Daniel M. Germán, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, September 20-24, 2010*, pages 437–446, 2010.

[107] Robert Gobeille. The FOSSology project. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings*, pages 47–50, 2008.

[108] Robert J. Grissom and John J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2nd edition edition, 2005.

[109] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[110] John Hartsock. jquery, jquery ui, and dual licensed plugins (dual licensing) [closed] `http://stackoverflow.com/questions/2758409/jquery-jquery-ui-and-dual-licensed-plugins-dual-licensing`. Last accessed: 2015/02/15.

[111] Sture Holm. A simple sequentially rejective Bonferroni test procedure. *Scandinavian Journal on Statistics*, 6:65–70, 1979.

[112] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6(5):429–449, October 2002.

[113] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, 2014.

[114] Camelia Lemnaru and Rodica Potolea. *Enterprise Information Systems: 13th International Conference, ICEIS 2011, Beijing, China, June 8-11, 2011, Revised Selected Papers*, chapter Imbalanced Classification Problems: Systematic Study, Issues and Best Practices, pages 35–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[115] Libraries.io. Libraries.io open data `https://libraries.io/data`, 2018.

[116] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. ChangeScribe: A tool for automatically generating commit messages. In *37th IEEE/ACM International Conference on Software Engineering (ICSE'15), Formal Research Tool Demonstration*, pages 709–712, 05 2015.

[117] Mario Linares-Vásquez, Kamal Hossen, Hoang Dang, Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. Triaging incoming change requests:

Bug or commit history, or code authorship? In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 451–460, Sept 2012.

[118] MARIO LINARES-VÁSQUEZ, COLLIN MCMILLAN, DENYS POSHYVANYK, AND MARK GRECHANIK. On using machine learning to automatically classify software applications into domain categories. *Empirical Softw. Engg.*, 19(3):582–618, June 2014.

[119] YUKI MANABE, YASUHIRO HAYASE, AND KATSURO INOUE. 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), Antwerp, Belgium, September 20-21, 2010*, pages 83–87. ACM, 2010.

[120] COLLIN MCMILLAN, MARK GRECHANIK, AND DENYS POSHYVANYK. Detecting similar software applications. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 364–374, Piscataway, NJ, USA, 2012. IEEE Press.

[121] COLLIN MCMILLAN, MARIO LINARES-VÁSQUEZ, DENYS POSHYVANYK, AND MARK GRECHANIK. Categorizing software applications for maintenance. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 343–352, Sept 2011.

[122] TIM MENZIES AND ANDRIAN MARCUS. Automated severity assessment of software defect reports. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 346–355, Sept 2008.

[123] NANCY J. MERTZEL. Copying 0.03 percent of software code base not 'de minimis'. *Journal of Intellectual Property Law & Practice*, 3(9):547, 2008.

[124] MATTHEW B. MILES AND A. MICHAEL HUBERMAN. *Qualitative Data Analysis: A Sourcebook of New Methods*. Sage, Beverly Hills, CA, 1984.

[125] Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. On the detection of licenses violations in the android ecosystem. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 382–392, March 2016.

[126] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, pages 484–495, 2014.

[127] Oracle. MySQL - FOSS License Exception. `http://www.mysql.com/about/legal/licensing/foss-exception/`. Last accessed: 2015/01/15.

[128] Oracle. Mysql connectors. `http://dev.mysql.com/downloads/connector/`.

[129] Massimiliano Di Penta and Daniel M. Germán. Who are source code contributors and how do they change? In *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, pages 11–20, 2009.

[130] Simon Phipps. Github needs to take open source seriously `http://www.infoworld.com/d/open-source-software/github-needs-take-open-source-seriously-208046`.

[131] Joseph J Pollock and Antonio Zamora. Collection and characterization of spelling errors in scientific and scholarly text. *Journal of the American Society for Information Science*, 34(1):51–58, 1983.

[132] Germán Poo-Caamaño and Daniel M. German. The Right to a Contribution: An Exploratory Survey on How Organizations Address It. In *11th International Conference on Open Source Systems (OSS)*, volume AICT-451 of *Open Source Sys-*

tems: Adoption and Impact, pages 157–167, May 2015. Part 5: Intellectual Property and Legal Issues.

[133] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[134] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[135] Param Singh and Corey Phelps. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, 24(3):539–560, 2009.

[136] Manuel Sojer, Oliver Alexy, Sven Kleinknecht, and Joachim Henkel. Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code. *J. of Management Information Systems*, 31(3):287–325, 2014.

[137] Manuel Sojer and Joachim Henkel. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems*, 11(12):868–901, 2010.

[138] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427 – 437, 2009.

[139] Donna Spencer. *Card sorting: Designing usable categories.* Rosenfeld Media, 2009.

[140] Jack T. Confusion about dual license (mit/gpl) javascript for use on my website `http://programmers.stackexchange.com/questions/139663/confusion-about-dual-license-mit-gpl-javascript-for-use-on-my-website`. Last accessed: 2015/02/15.

[141] The Free Software Foundation. Gnu general public license, version 2, with the classpath exception http://openjdk.java.net/legal/gplv2+ce.html.

[142] The Linux Foundation. Software package data exchange - fltk exception. https://spdx.org/licenses/FLTK-exception.html.

[143] The Linux Foundation. Software package data exchange - license exceptions. https://spdx.org/licenses/exceptions-index.html.

[144] The Open Source Initiative. Open source licenses. http://opensource.org/licenses/category.

[145] The Open Source Initiative. Report of license proliferation committee and draft faq. http://opensource.org/proliferation-report.

[146] Timo Tuunanen, Jussi Koskinen, and Tommi Kärkkäinen. Automated software license analysis. *Autom. Softw. Eng.*, 16(3-4):455–490, 2009.

[147] Secil Ugurel, Robert Krovetz, and C. Lee Giles. What's the code?: Automatic classification of source code archives. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 632–638, New York, NY, USA, 2002. ACM.

[148] Sam Varghese. Linux developer loses gpl suit against vmware http://www.itwire.com/open-source/74288-linux-developer-loses-gpl-suit-against-vmware.html, 2017.

[149] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. License usage and changes: a large-scale study on github. *Empirical Software Engineering*, 22(3):1537–1577, Jun 2017.

[150] Christopher Vendome, Daniel German, Massimiliano Di Penta, Gabriele Bavota, Mario Linares-Vásquez, and Denys Poshyvanyk. To

distribute or not to distribute: Why licensing bugs matter. In *40th IEEE/ACM International Conference on Software Engineering (ICSE'18)*, 2018.

[151] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. Machine learning-based detection of open source license exceptions. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 118–129, 2017.

[152] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. Germán, and Denys Poshyvanyk. License usage and changes: A large-scale study of Java projects on GitHub. In *The 23rd IEEE International Conference on Program Comprehension, ICPC 2015, Florence, Italy, May 18-19, 2015*, pages 31–40. IEEE, 2015.

[153] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. When and why developers adopt and change software licenses. In *The 31st IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, pages 31–40. IEEE, 2015.

[154] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. Germán, and Katsuro Inoue. A method to detect license inconsistencies in large-scale open source projects. In *The 12th Working Conference on Mining Software Repositories MSR 2015, Florence, Italy, May 16-17, 2015*. IEEE, 2015.