Noname manuscript No.
(will be inserted by the editor)

License Usage and Changes: A Large-Scale Study on GitHub

Christopher Vendome,
Gabriele Bavota,
Massimiliano Di Penta,
Mario Linares-Vásquez,
Daniel German,
Denys Poshyvanyk

Received: date / Accepted: date

Abstract Open source software licenses determine, from a legal point of view, under which conditions software can be integrated and redistributed. The reason why developers of a project adopt (or change) a license may depend on various factors, *e.g.,* the need for ensuring compatibility with certain third-party components, the perspective towards redistribution or commercialization of the software, or the need for protecting against somebody else's commercial usage of the software. This paper reports a large empirical study aimed at *quantitatively* and *qualitatively* investigating when and why developers adopt or change software licenses.

Specifically, we first identify license changes in 1,731,828 commits, representing the entire history of 16,221 Java projects hosted on GitHub. Then, to understand the rationale of license changes, we perform a qualitative analysis on 1,160 projects written in seven different programming languages, namely C, C++, C#, Java, Javascript, Python, and Ruby —following an open coding approach inspired

C. Vendome
The College of William and Mary
E-mail: cvendome@cs.wm.edu

G. Bavota
Free University of Bozen-Bolzano
E-mail: gabriele.bavota@unibz.it

M. Di Penta
University of Sannio
E-mail: dipenta@unisannio.it

M. Linares-Vásquez
The College of William and Mary
E-mail: mlinarev@cs.wm.edu

D. M. German
University of Victoria
E-mail: dmg@cs.uvic.ca

D. Poshyvanyk
The College of William and Mary
E-mail: denys@cs.wm.edu

by grounded theory—on commit messages and issue tracker discussions concerning licensing topics, and whenever possible, try to build traceability links between discussions and changes. On one hand, our results highlight how, in different contexts, license adoption or changes can be triggered by various reasons. On the other hand, the results also highlight a lack of traceability of *when* and *why* licensing changes are made. This can be a major concern, because a change in the license of a system can negatively impact those that reuse it. In conclusion, results of the study trigger the need for better tool support in guiding developers in choosing/changing licenses and in keeping track of the rationale of license changes.

Keywords Software Licenses · Mining Software Repositories · Empirical Studies

1 Introduction

In recent and past years, the diffusion of Free and Open Source Software (FOSS) projects is increasing significantly, along with the availability of forges hosting such projects (*e.g.,* SourceForge[1] or GitHub[2]) and foundations supporting and promoting the development and diffusion of FOSS (*e.g.,* the Apache Software Foundation[3], the GNU Software Foundation[4], or the Eclipse Software Foundation[5]). The availability of FOSS projects is a precious resource for developers, who can reuse existing assets, extend/evolve them, and in this way create new work productively and reduce costs. For example, a blog post by IBM[6] outlines the reasons pushing companies to reuse open source code: "Yes, this [the cost factor] is one of the most important factors that attract not only the small companies or start-up's but also the big corporations these days". This can happen not only in the context of open source projects, but it is more and more frequent in commercial projects. In a survey conducted by Black Duck[7], it was found that 78% of the companies use open source code (double from 2010), 93% claimed an increase in open source reuse, 64% contribute to open source development, and over 55% indicated a lack of formal guidance when utilizing open source code. The findings by Black Duck demonstrate two key implications: i) commercial reuse of open source code has been increasing, and ii) in general, there is a lack of oversight in how this reuse occurs.

Nevertheless, whoever is interested in integrating FOSS code in their software project (and redistributing along with the project itself), or modifying existing FOSS projects to create a new work—referred to as "derivative work"—must be aware that such activities are regulated by *software licenses* and in particular by the specific

[1] http://sourceforge.net

[2] https://github.com

[3] https://www.apache.org

[4] http://www.gnu.org

[5] http://www.eclipse.org/

[6] https://www.ibm.com/developerworks/community/blogs/6e6f6d1b-95c3-46df-8a26-b7efd8ee4b57/entry/why_big_companies_are_embracing_open_source119?lang=en

[7] https://www.blackducksoftware.com/future-of-open-source

FOSS license of the project being reused. In order to license software projects, developers either add a *licensing statement* to source code files (as a comment at the beginning of each file) and/or include a textual file containing the license statement in the project source code root directory or in its sub-directories.

Generally speaking, FOSS licenses can be classified into *restrictive* (also referred to as "copyleft" or "reciprocal") and *permissive* licenses. A restrictive license requires developers to use the same license to distribute new software that incorporates software licensed under such restrictive license (*i.e.,* the redistribution of the derivative work must be licensed under the same terms); meanwhile, permissive licenses allow re-distributors to incorporate the reused software under a difference license [74, 51]. The *GPL* (in all of its versions) is a classic example of a restrictive license. In Section 5 of the *GPL-3.0*, the license addresses code modification stating that "*You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy*" [11]. The *BSD* licenses are examples of permissive licenses. For instance, the *BSD 2-Clause* has two clauses that detail the use, redistribution, and modification of licensed code: (i) the source must contain the copyright notice and (ii) the binary must produce the copyright notice and contain the disclaimer in documentation [4].

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.* [46], the choice of the licenses in a FOSS 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 [23]—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 [70] 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 FOSS 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 [56, 55] and identifying licensing incompatibilities [52]. Even initiatives such as the Software Package Data Exchange (SPDX) [30] have been aimed at proposing a formal model to document the license of a system. However, despite of the effort put by the FOSS 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 [78]).

Paper contributions. This paper 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 FOSS 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 [42], 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.

The study reported in this paper poses its basis on previous work aimed at exploring license incompatibilities [52], license changes [46], license evolution[61], and integration patterns [54]. Building upon previous work on licensing analysis, this paper:

1. Constitutes, to the best of the authors' knowledge, the largest study aimed at analyzing the change patterns in licensing of software systems (earlier work was limited to the analysis of up to six projects [61,46]).
2. To the best of our knowledge, it is the first work aimed at explaining the rationale of license changes by means of a qualitative analysis of commit notes and issue tracker discussions.

The achieved results 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. For example, during the creation of a new repository, GitHub allows the user to select an initial license from a list of commonly used ones, but offers no guidance on the implications of such a choice, and simply redirects the user to `http://choosealicense.com/`; aside from this, GitHub offers no support for licensing management. Also, there is a lack of consistency and standardization in the mechanism that should be used for declaring a license (*e.g.,* putting it in source code

heading comments, separate license files, README files, *etc.*). Moreover, the legal nature of the licenses exacerbate this problem since the implications and grants or restrictions are not always clear for developers when the license is present. Last, but not least, the currently available Software Configuration Management (SCM) technology provides no support to trace licensing-related discussions and decisions onto actual changes, whereas such traceability links can be useful to understand the impact of such decisions.

Paper structure. The paper is organized as follows. Section 2 relates this work to the existing literature on licensing analysis. Section 3 describes the study design and details the data analysis procedure. Results are reported and discussed in Section 4. Lessons learned from the study results are summarized in Section 5, while Section 6 discusses the threats to the study's validity. Finally, Section 7 concludes the paper and outlines directions for future work.

2 Related Work

Our work is mainly related to (i) techniques and tools for automatically identifying and classifying licenses in software artifacts, and (ii) empirical studies focusing on different aspects of license adoption and evolution.

2.1 Identifying and Classifying Software Licenses

The problem of license identification has firstly been tackled in the FOSSology project [56] aimed at building a repository storing FOSS projects and their licensing information and using a machine learning approach to classify licenses. Tuunanen *et al.* [76] proposed ASLA, a tool aimed at identifying licenses in FOSS systems; the tool has been shown to determine licenses in files with 89% accuracy.

German *et al.* [55] proposed Ninka, a tool that uses a pattern-matching based approach for identifying statements that characterize various licenses. Given any text file as an input, Ninka outputs the license name and version. In the evaluation reported by the authors, Ninka achieved a precision $\sim$95% while detecting licenses. Ninka is currently considered the state-of-the-art tool in the automatic identification of software licenses.

While the typical license classification problem arises when source code is available, in some cases, source code is not available—*i.e.,* only byte code or binaries are available—and the goal is to identify whether the byte code has been produced from source code under a certain license. To this aim, Di Penta *et al.* [45] combined code search and textual analysis to automatically determine a license under which jar files were released. Their approach automatically infers the license from decompiled code by relying on the Google Code search engine. Note that, differently from the previous techniques, the approach in [45] is only able to identify the license family (*e.g.,* GPL) without specifying the version (*e.g.,* 2.0).

Di Penta *et al.* [46] investigated—on six open source projects written in C, C++ and Java—the migration of licenses over the course of a project's lifetime. The study suggests that licenses changed version and type during software evolution, but there was no generic patterns generalizable to the six analyzed FOSS projects. Also, Manabe *et al.* [62] analyzed the changes in licenses but of FreeBSD, OpenBSD, Eclipse, and ArgoUML, finding that each project had different evolution patterns.

German *et al.* [54] 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.* [52] 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; they were able to find some licensing issues confirmed by Fedora.

German *et al.* [53] analyzed the presence of cloned code fragments between the Linux Kernel and two distributions of BSD, *i.e.,* OpenBSD and FreeBSD. The aim was to verify whether the cloning was performed in accordance to the terms of the licenses. Results show that, in most cases, these code-migrations were admitted since they went from less restrictive licenses towards more restrictive ones.

Wu *et al.* [79] investigated license inconsistencies between cloned files. They performed an empirical study on Debian 7.5 to demonstrate the ways in which licensing can become inconsistent between the file clones (*e.g.,* the removal of a license in one of the clone pairs).

In our previous work [77], we focused our analysis only on Java projects. In this work, we expand our analysis to include six new languages—C, C++, C#, Javascript, Python, and Ruby. Also, our new grounded theory analysis features a categorization of commit messages and issue discussions into seven categories, in turn further detailed in a total of 27 sub-categories. In addition to extracting new support and rationale, we also defined new sub-categories and subsequently distilled lessons from this new data. For example, we observed that asserting a license is not standardized or consistent across languages, and it would benefit developers to have a consistent means of documenting and presenting the license of a system within a forge.

Vendome *et al.* [78] conducted a survey with developers that contributed to projects that had experienced changes in licensing to understand the rationale for adopting and changing licensing. The survey results indicated that facilitating commercial reuse is a common reason for license changes. Also the survey highlighted that, in general, developers have a lack of understanding of the legal implications of open source licenses, highlighting the need for recommenders aimed at supporting them in choosing and changing licenses.

While we share similar goals with prior related work—understanding insights into license usage and migration—our analysis is done on a much larger scale, including a (i) quantitative analysis on 16,221 Java projects, and (ii) a qualitative analysis upon a sample of commit messages and issue tracker discussions from 1,160 projects

written in seven different programming languages. The latter allowed us to perform in-depth analysis of the rationale behind license usages and migrations.

3 Design of the Empirical Study

The *goal* of our study is to investigate license adoption and evolution in FOSS projects, with the *purpose* of understanding the overall rationale behind picking a particular license or changing licenses and of determining the underlying license change patterns. The *perspective* is of researchers interested in understanding what are the main factors leading towards specific license adoption and change. The *context* consists of (i) the change history of 16,221 Java open source projects mined from GitHub, which will be used to quantitatively investigate the goals of the study, and (ii) commit messages and issue tracker discussions from 1,160 projects written in seven different programming languages (*i.e.,* C, C++, C#, Java, JavaScript, Python, and Ruby), which will be exploited for qualitative analysis.

3.1 Research Questions

We aim at answering the following research questions:

1. **RQ**$_1$ *What is the usage of different licenses by projects in GitHub?* This research question examines the proportions of different types of licenses that are introduced by FOSS projects hosted in GitHub. In doing this, we should consider that GitHub is a relatively young forge (launched in April 2008), which has seen exponential growth in the number of projects over the past few years, and that most of the projects it hosts are young in terms of the first available commit or the date that the repository was created.
2. **RQ**$_2$ *What are the most common licensing change patterns?* Our second research question investigates the popular licensing change patterns in the GitHub Open Source community with the aim of driving out—from a qualitative point of view—the rationale behind such change patterns (*e.g.,* satisfying dependency constraints).
3. **RQ**$_3$ *To what extent are licensing changes documented in commit messages or issue tracker discussions?* This research question investigates on whether licensing changes in a system can be traced to commit messages or issues' discussions.
4. **RQ**$_4$ *What rationale do these sources contain for the licensing changes?* This research question investigates the rationale behind the particular change in license(s) from a developer's perspective.

We address our four research questions by looking at the licensing phenomenon from two different points of view, namely (i) a *quantitative* analysis of the licenses under which projects were released, their changes across their evolution history, and the ability to match these changes to either commit messages or issue tracker discussions; and (ii) a *qualitative* analysis of licensing-related discussions made by developers over the issue trackers and of the way in which developers documented

licensing changes through commit messages. For the quantitative analysis of licensing changes, we are interested in analyzing license migration patterns that fall in the following three categories:

- *No license* $\rightarrow$ *some License(s)* – $N2L$. This reflects the case in which developers realized the need for a license and added a licensing statement to files;
- *some License(s)* $\rightarrow$ *No license* – $L2N$. In this case, for various reasons, licensing statements have been removed from source code files; for example, because a developer accidentally added a wrong license/license version;
- *some License(s)* $\rightarrow$ *some other License(s)* – $L2L$. This is the most general case of a change in licensing between distinct licenses.

To address $\mathbf{RQ}_1$, $\mathbf{RQ}_2$, and $\mathbf{RQ}_3$, we perform a quantitative analysis by mining the version history of 16,221 Java projects, while to address $\mathbf{RQ}_4$ we perform a qualitative analysis on the commit messages and issue tracker discussion of the 1,160 projects written in seven different programming languages. In the following subsections, we describe the two kinds of analysis in detail.

3.2 Quantitative Analysis

In order to generate the dataset to be used in the study, we mined the version history of 16,221 Java projects publicly available on GitHub. GitHub hosts over twelve million *Git* repositories covering many popular programming languages, and provides a public API [10] that can be used to query and mine project information. Also, the *Git* version control system allows for local cloning of the entire repository, which facilitates the comprehensive analysis of the project change-history and thus of the license changes happened in each commit.

To extract data for our quantitative analysis, we first identified a comprehensive list of projects hosted on GitHub by implementing a script exploiting GitHub's APIs. The computation of the comprehensive list resulted in over twelve million projects. Since the infrastructure we use for license extraction only supports Java systems (as it will be explained later), we filtered out all systems that were not written in Java, obtaining a list of 381,161 Java projects hosted on GitHub. We cloned all 381,161 git repositories locally for a total of 6.3 Terabytes of storage space. In our analysis, we randomly sampled 16,221 projects due to the computation time of the aforementioned infrastructure.

Once the *Git* repositories had been cloned, we used a code analyzer developed in the context of the MARKOS European project [39] to extract license information at commit-level granularity. The MARKOS code analyzer uses the Ninka license classifier [55] to identify and classify licenses contained in all the files hosted under the version control system of each project. For each of the 16,221 projects in our study, the MARKOS code analyzer mined the change log, producing the following information for each commit:

1. *Commit Id:* The identifier of the commit that is currently checked out from the Git repository and analyzed;
2. *Date:* The timestamp associated with the commit;

3. *Author:* The person responsible for the commit;
4. *Commit Message:* The message attached to the commit;
5. *File:* The path of the files committed;
6. *Change to File:* A field to indicate whether each file involved in the commit was Added, Deleted, or Modified;
7. *License Changed:* A boolean value indicating whether the particular file has experienced a change in license in this commit with respect to its previous version. This feature applies to modified files only. In the case of an addition or deletion of a file, this field is set to *false*;
8. *License:* The name and version (*e.g., GPL-2.0*) of each license applied to the file.

The computation of such information for all 16,221 projects took almost 40 days, and resulted in the analysis of a total of 1,731,828 developers' commits involving 4,665,611 files. Note that for the BSD and CMU licenses Ninka was not able to correctly identify its variants (reporting it as *BSD var* and *CMU var*). Additionally, the GPL and the LGPL may contain a "+" after the version number (*e.g.,* 3.0+), which represents a clause in the license granting the ability to use future versions of the license (*i.e.,* the *GPL-2.0+* would allow for utilization under the terms of the *GPL-3.0*). Also, we have values of "no license" and "unknown", which represents the case that no license was attached to the file or Ninka was unable the determine the license.

To determine whether there is a trend in the proportions of adopted licenses over the observed years, we used the Augmented Dickey-Fuller (ADF) test [47,48]. This test is widely used to test stationarity of time series. The test can be used to reject two different null hypotheses H_{0s}*: the time series is not significantly stationary* or H_{0e}*: the time series is not significantly explosive*; the latter can be used to determine whether there is a significantly increasing trend in the time series. In our statistical tests, we considered a significance level of 0.05 (*i.e.,* we rejected null hypotheses for p-values < 0.05).

We quantitatively analyzed the collected data by presenting descriptive statistics about the license adoption and the most common *atomic license changes* that we found. The latter are defined as the commits in which we detected a specific kind of license change within at least one source code or textual file. For example, given a commit with three files experiencing the licensing change *No license* $\rightarrow$ *Apache-2.0*, and 10 files with *GPL-2.0* $\rightarrow$ *GPL-3.0*, the atomic license changes from that commit are one *No License* $\rightarrow$ *Apache-2.0* change and one *GPL-2.0* $\rightarrow$ *GPL-3.0* change. We prefer not to count the number of changes at file level as it was done in previous work [46] to avoid inflating our analysis because of large commits and to make comparable commits performed on both small and large projects. It is possible that this coarse-grained analysis may fail to capture some license changes, for example due to a change in licensing of a dependency, although also in this case, in principle, the licensing changes should be reflected at project level when appropriate.

In the end, we identified a total of 1,833 projects with *atomic license changes* out of our dataset of 16,221 projects. This subset of projects was used to investigate license change traceability. Intuitively, we require the presence of license changes in order to determine how well changes in licensing are documented in either the

commit messages or issue tracker discussion. Therefore, we used a web crawler to identify, among these 1,833 projects, those using the GitHub issue tracker, finding a total of 1,586 projects having at least one issue on it. To link the licensing changes to commit messages/issue reports, we performed both string matching and date matching between either the commit messages or the issue tracker discussions and the extracted licensing information (*e.g.,* license name or date that license was committed). We decided to rely on commit messages and issue discussions because (i) these two sources of information are publicly available for the considered subject projects; and (ii) both commit messages and issue discussions are likely to report, with a different level of detail, the rationale behind a specific change implemented (or just considered in the case of issues) by developers, including changes related to software licenses.

3.3 Qualitative Analysis

Our qualitative analysis aims at answering $\mathbf{RQ}_4$ and it is based on manual inspection and categorization of commit messages and issue tracker discussions. Since we do not have limitations in terms of the project's programming language to analyze (unlike the quantitative analysis), we performed our qualitative analysis on commit messages and issue tracker discussions from a set of 1,160 projects written in seven different languages: 159 C, 91 C++, 78 C#, 324 Java, 166 Javascript, 147 Python, and 195 Ruby projects. Note that the choice of the languages considered in our study is not random: we focused on seven of the ten most popular programming languages during 2014 and 2015 [80, 41].

The considered projects were instead selected by applying the following procedure. Firstly, from our list of twelve million repositories, we extracted those written in the seven languages of interest. Then, we extracted only the repositories satisfying the following two criteria: (i) they were not forks of the main repository, and (ii) they had at least one star (*i.e.,* at least one user expressed appreciation for the repository) or watcher (*i.e.,* at least one user asked to receive notification about changes made in the repository). These selection criteria were used to exclude from our analysis personal repositories (*e.g.,* the website of a GitHub user) that might have biased our results. However, it is important to note that for Java, we considered the comprehensive list of all 381,161 projects. In our initial investigation of Java projects [77], we observed the need for refinement that was thus adopted for the additional six languages, because we observed a high proportion of false positive commit messages and issues discussions. Thus, the filtering sought to improve the generated taxonomy.

Then, we extracted the change log of the cloned projects in order to analyze them and identify the commit messages likely related to licensing. In total, 103,128,211 commits were considered. To identify commit messages likely related to license changes, we adopted a case-insensitive keyword-based filtering based on the critical words exploited by Ninka during license identification, and augmented them with license names. The detailed set of keywords being used for this matching is reported in Table 1. In some cases, our keyword-filters included bi-grams composed of the license type and version, since some license types (*e.g.,apache*) produced a very large

Table 1 List of keywords used to match candidate licensing-related commit messages and issue tracker discussions.

copyright, compliance, gpl, gpl-2, gpl-3, gplv2, gplv3, gplv2+, gplv3+, lgpl, lgpl-2, lgpl-2.1, lgpl-3, lgplv2, lgplv2.1, lgplv3, lgplv2+, lgplv2.1+, lgplv3+, licenses, license, licensed, licensee, lgpl, merchantability, mit/x-derivative, mpl, written permission, prior permission, see the copyright.txt, licensing, licencing, liability, legal, public domain, special exception, copyright holders, to permit this exception, disclaims copyright, gpl, apache-2, apache-2.0, apache 2, apache 2.0, apache v2, apache v2.0, apache-1.1, apache 1.1, apache v1.1, apl-1.1, apl-1.1, apl 1.1, apl v1.1, gpl 3, gpl 3+, gpl 2, gpl 2+, lgpl 2, lgpl 2+, lgpl 2.1, lgpl 2.1+, lgpl 3, lgpl 3+, gpl v3, gpl v3+, gpl v2, gpl v2+, lgpl v2, lgpl v2+, lgpl v2.1, lgpl v2.1+, lgpl v3, lgpl v3+, mit/x, mit/x11, mit x11, mit expat, cpl-1.0, cpl-1, epl-1.0, epl-1, cpl 1.0, cpl 1, epl 1.0, epl 1, cddl-1.0, cddl-1, cddl 1.0, cddl 1, cpl v1.0, cpl v1, epl v1.0, epl v1, cddl v1.0, cddl v1, mpl-1.0, mpl-2.0, mpl-1, mpl-2, mpl 1.0, mpl 2.0, mpl 1, mpl 2, mpl v1.0, mpl v2.0, mpl v1, mpl v2, bsd-3, bsd-2, bsd-4, bsd 3-clause, bsd 2-clause, bsd 4-clause

amount of false positive discussions when they were considered alone (*e.g.,* all the commit message talking about Apache projects).

In the end, the keyword-based filtering allowed us to identify a total of 746,874 commit messages (742,671 for Java, which amounted to approximately $\sim$1% of the overall commits for Java). Given the high number of relevant commits, we sampled 20% of the commits found for each language as object of our manual inspection. However, we set a minimum threshold of 100 commits per language, and a maximum threshold of 500. These thresholds were adopted to ensure representativeness for each of the studied language, while keeping the manual analysis effort reasonable. Note that our sampling is statistically significant with a 95% confidence interval $\pm$ 10% or better. This resulted in a total of 1,413 commits to be inspected. It is worth noting that for Java projects, in addition to the 500 sampled commit messages matching the keywords in Table 1, we also considered 224 randomly sampled commit messages from the commits of the 1,833 projects in which we identified (in our quantitative analysis) an instance of an *atomic license change*, because we were interested in investigating the reasons behind such changes. Clearly, this was not possible for the systems written in other programming languages that, as said before, were not part of our quantitative analysis. The number of sampled commits by each programming language is reported in the second column of Table 2.

Concerning the issue tracker discussions, we built a Web crawler collecting the information present in all issue trackers of the studied projects. In particular, for each issue, our crawler collected (i) its title and description, (ii) the text of each comment added to it, (iii) and the date the issue was opened and closed (when applicable). Then, in order to find the relevant issues (*i.e.,* those presenting discussions about software licenses), we used a keyword search mechanism aimed at matching, in the issue title, keywords related to licensing (as previously explained for the commit messages)[8]. By applying this procedure, we identified a total of 486 issue discussions potentially related to licensing, as shown in the third column of the Table 2.

After collecting commit messages and issue discussions, in order to analyze and categorize them, we followed an open coding process inspired by the Grounded The-

[8] We looked for the target keywords only in the issue titles, because we found that including the issue descriptions in the search generates a considerable number of false positives.

Table 2 The number of commits and issue tracker discussions considered in the qualitative analysis.

Language	#of commits	# of issue tracker discussions
C	227	30
C#	100	6
C++	139	12
Python	130	41
Java	724	273
JavaScript	122	79
Ruby	195	45
Overall	**1,637**	**486**

ory (GT) principles formulated by Corbin and Strauss [42]. This analysis of commit messages and issue tracker discussions aimed at finding the rationale for licensing changes; in particular, we aimed at answering the following two sub-questions: *What are the reasons pushing developers to associate a particular license to their project?* and *What causes them to migrate licenses or release their project under a new license (i.e., co-licensing)?*

To perform the open coding, we distributed the commit messages and the issue tracker discussions among the authors such that two authors were randomly assigned to each message (a message can be a commit message or an entire issue tracker discussion). After each round of *open coding* in which the authors independently created classifications for the messages, we met to discuss the coding identified by each of us, and we refined them into categories. Note that during each round the categories defined in previous rounds were refined accordingly to the new knowledge created from the additional manual inspections and from the authors' discussions. Overall, the open coding concerned (i) 1,413 randomly selected licensing-related commit messages identified via the keywords-based mechanism; (ii) the 224 commit messages from the Java systems' commits where a licensing change was observed in our quantitative analysis; and (iii) the 486 issue tracker discussions matching licensing-related keywords. The output of our open coding procedure is a set of categories and group explaining why licenses are adopted and changed. We qualitatively discuss the findings of this analysis in Section 4.4, presenting our categories classification and examples of commit messages and issue tracker discussions belonging to the various categories.

3.4 Dataset Diversity Analysis

To get an idea of the external validity of our dataset, we measured the diversity metric proposed by Nagappan *et al.* [69] for our dataset by matching the list of our mined projects from GitHub to the list of available projects from Boa [50]. Given the different datasets exploited in the context of our quantitative and qualitative analysis, we discuss the diversity metrics separately.

Table 3 Top licenses: OSI, SourceForge, and our dataset.

OSI Popular License (unordered)	SourceForge (Dec. 2009)	Our Github Data Set (Quant. Analys.)
Apache-2 Lic	GNU Public Lics	GNU Public Lics
BSD 2-Clause Lic	Lesser GNU Public Lics	Apache Lics
BSD 3-Clause Lic	BSD Lics	Lesser GNU Public Lics
GNU Public Lics	Apache Lics	MIT Lic
Lesser GNU Public Lics	Public Domain	Eclipse Public Lic
MIT Lic	MIT Lic	Comm. Dev. and Dist. Lic
Mozilla Public Lic 2	Academic Free Lic	Mozilla Public Lic
Comm. Dev. and Dist. Lic	Mozilla Public Lics	BSD Lics
Eclipse Public Lic		

Table 4 Projects in our dataset with an initial commit for each year.

Year	Projects	Year	Projects	Year	Projects	Year	Projects	Year	Projects
1992	1	2000	7	2004	22	2008	186	2012	14159
1996	1	2001	11	2005	36	2009	263	2013	60
1997	3	2002	10	2006	72	2010	440		
1999	6	2003	35	2007	91	2011	811		

3.4.1 Quantitative Analysis

We were able to match by name 1,556 out of the 16,221 projects exploited in our quantitative analysis against the names of the projects in the diversity metric dataset by Nagappan *et al.* [69]. This subset was used in the computation of the diversity metric, obtaining a score of 0.35, indicating that around 10% of our dataset covers just over a third of the open source projects according to six dimensions: programming language, developers, project age, number of committers, number of revisions, and number of programming languages. The dimensional scores are 0.45, 0.99, 1.00, 0.99, 0.96, 0.99, respectively, suggesting that our subset covers the relevant dimensions for our analysis. However, the focus on Java projects limits the programming language score, affecting the overall score.

Another important aspect to evaluate is the representativeness of the licenses present in our dataset with respect to those diffused in the FOSS community. The Open Source Initiative (OSI) specifies a list of 70 approved licenses, indicating the ones reported in the first column of Table 3 as the most commonly used in FOSS software (they do not specify any order). The second column of Table 3 reports the top licenses as extracted from the FLOSSmole's SourceForge snapshot of December 2009 [59], while the third column shows the top licenses as extracted from our sample of GitHub projects exploited for the quantitative analysis.

The licenses declared by OSI as the most commonly used were also the most commonly found in our dataset (BSD 2 and 3 fall both in the BSD type). In the comparison between our dataset and SourceForge, while the order of diffusion for the different licenses is not exactly the same, six of the top eight licenses in SourceForge are also present in our dataset (all but Public Domain and Academic Free License). This analysis, together with the diversity metric, suggests that the dataset we exploited in our quantitative analysis is representative of Open Source systems.

Table 4 reports the year of the first commit date for each of the 16,221 considered projects. This table clearly shows the exponential growth of GitHub until 2012, confirming what already was observed by people in the GitHub community [49]. While GitHub also experienced exponential growth in 2013 [31], our dataset does not mirror this fact. This is due to a design choice we made while randomly choosing the projects to clone. In particular, we cloned projects during January 2014, excluding projects with a commit history less than one year from the set of 381,161 Java projects (*i.e.,* projects with the first commit performed no later than January 2013). This was needed since, in the context of $\mathbf{RQ}_2$, we are interested in observing migration patterns occurring over the projects' history. Thus, projects having a very short commit history were not likely to be relevant for the purpose of this study. Moreover, since in $\mathbf{RQ}_1$ we are interested in observing licenses' usage in the context of the GitHub's drastic expansion, we decided to exclude the 60 projects having the first commit in 2013 from our analysis due to the severe lack of representation in our sample despite the continued growth of GitHub.

3.4.2 Qualitative Analysis

Similarly, we were able to match by name 471 out of the 1,160 projects (against the names of the projects in the diversity metric dataset [69]) from which we manually investigated commit messages and issue discussions in our qualitative analysis. As done for the quantitative analysis, we considered the matched subset for the computation of the diversity metric, obtaining a score of 0.32, indicating that $\sim$40% of our dataset covers just under a third of the open source projects according to six dimensions: programming language, developers, project age, number of committers, number of revisions, and number of programming languages. The dimensional scores are 0.43, 0.99, 1.00, 0.99, 0.94, and 1.0, respectively. Intuitively, these scores are directly impacted by the limited number of projects that we were able to match. However, we still observe relatively high diversity scores suggesting that our qualitative analysis is representative for a substantial portion of the open source systems.

3.5 Replication Package

The working data set of our study is available at: `http://www.cs.wm.edu/semeru/data/EMSE15-licensing`. It includes (i) the lists of projects and their urls, (ii) the issues tracker and commit data, (iii) the analysis scripts, and (iv) a summary of the achieved results.

4 Study Results

This section discusses the achieved results answering the four research questions formulated in Section 3.1.

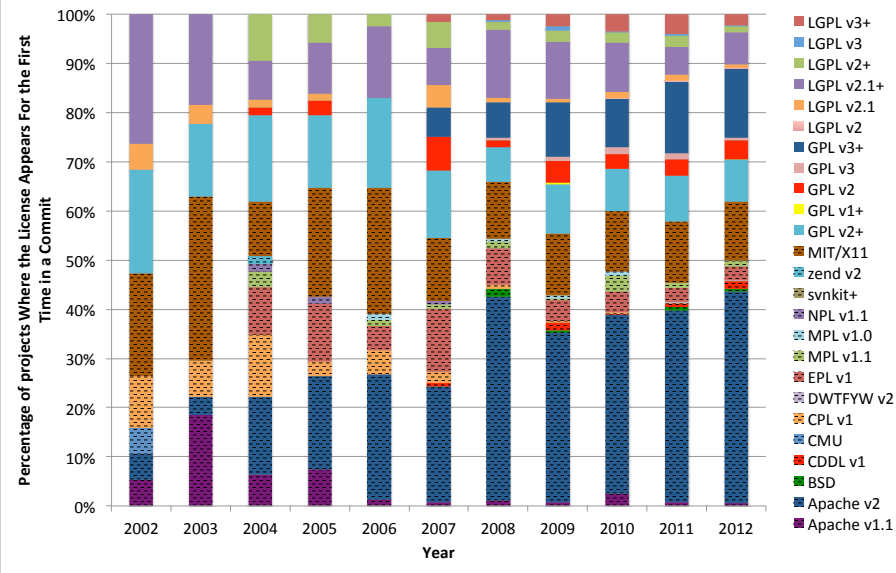

Fig. 1 Relative license usage of the analyzed Java projects between 2002 and 2012 (dashed pattern representing permissive licenses).

4.1 RQ_1: What is the usage of different licenses in GitHub?

Fig. 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 (see Table 4).

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 the *MIT/X11* 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 ob-

servation period, Bavota *et al.* found that the Apache ecosystem grew exponentially [38]. This observation explains the rapid diffusion of the *Apache-2.0* license among FOSS projects. We observed a growth that resulted in the *Apache-2.0* license accounting for approximately 41% of licensing in 2008. Conversely, we observed a decline in the relative usage of both *GPL* and *LGPL* licenses. These two observations suggest a clear shift toward permissive licenses, since ∼60% of licenses attributed were permissive starting from 2003 (with small drops in 2007 and 2009).

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 FOSS 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/X11* 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/X11* usage at approximately 10% starting in 2007.

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 as explained in Section 3.2. Results are reported in Table 5, 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+*.

Table 5 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/X11	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

– Confirming the discussion above, we have a clearly increasing trend not only for permissive licenses such as *Apache-2.0* and MIT/X11 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, as it will be discussed in Section 5, 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 a clear trend towards using permissive licenses like *Apache-2.0* and *MIT/X11*. 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.

Table 6 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/X11*	163
no license or unknown → *GPL-2.0+*	113
GPL-2.0+ → no license or unknown	111
MIT/X11 → 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/X11*	11
MIT/X11 → *Apache-2.0*	11

4.2 RQ_2: What are the most common licensing change patterns?

We analyzed commits, where a license change occurred, 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 6) and dominant local patterns (Table 7) 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 6 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 6 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

Table 7 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

license upgrade and downgrade between *GPL-2.0+* and *GPL-3.0+*. *GPL-3.0* is considered by the Free Software Foundation as a compatible license with the *Apache-2.0* license[9]. 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 7), 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 7 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) 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 8 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* → *MIT/X11* was the most prevalent license change pattern, and the co-license of *MIT/X11* 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/X11* license

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

Table 8 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 → *MIT/X11*	12
Apache-2.0 → *MIT/X11*;*Apache-2.0*	8
GPL-2.0+ → *GPL-3.0+*	7
MIT/X11 → *Apache-2.0*	6
GPL-3.0+ → *Apache-2.0*	6
MIT/X11;*Apache-2.0* → *Apache-2.0*	5
Apache-2.0 → *GPL-3.0+*	5
GPL-3.0+ → *MIT/X11*	3

was used to allow commercial reuse, while still maintaining the open source nature of the project.

The pattern of *GPL-2.0+* → *GPL-3.0+* (Top-3 in Table 8) was expected since it was tied for the most prevalent among global *atomic license changes*. Similarly, the patterns of *MIT/X* → *Apache-2.0*, *GPL-3.0+* → *Apache-2.0*, and *Apache-2.0* → *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* → *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 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.

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

Table 9 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.

Table 9 Traceability between licensing changes and commit messages or issue tracker discussion comments.

Data Source	Linking Query	Links
Commit Messages	Commits with the keyword "license"	70,746
	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	2,241
	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]"	66,025
	Comments in open issues containing the keyword "Fixed #[issue_num]"	3,407
	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

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.

4.4 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. As explained in Section 3, 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

Table 10 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 FOR A LICENSE																
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**

findings when looking at the commits that introduce *atomic license changes* in the analyzed Java projects.

4.4.1 Analyzing Commit Messages and Issue Discussions

Table 10 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 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 fil-

tering (see Section 3) 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." [35]

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." [29]

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" [36] in the project *webworkshops*, 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?" [7]

Interestingly, one developer submitted an issue for the project *InTeX* entitled "Dual license under LGPL and EPL" [15] 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[10]. 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:

> *"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* [6] 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

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

changes applied in the repository to ensure compliance to the new licensing terms (*e.g.,* the removal of the ztree class). 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* [20]:

> *"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 [24]. 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"* [28]

> *"Remove extra LICENSE files*
> *One repository, one license. No need to put these on the box either."* [8]

> *"Remove licenses for unused libraries"* [5]

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 [71]. 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 [46].

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 *gtksourcecompletion*'s issue tracker [12]:

> *"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" [33] 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* [27]:

> *"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* [14]:

> *"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* [32]. 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."* [18]

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"* [37]. 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* [13]:

> *"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* [2], 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 [9], a well-known forge of 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* [26] 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 all copyright holders 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* [22] 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 [78].

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* [34] 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* [1] 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?"* [25] 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 [19] 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.

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 [3]. Similarly, in our previous Java study [77], a developer submitted a patch, but it could not be merged into the system until that developer filled out the CLA [16]. 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 [40]. Thus, it prevents the contributed code from becoming a ground for a potential lawsuit.

Request for a license. 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* [17] 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 [77, 78, 75] 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)." [21]

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.

4.4.2 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 11. 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"

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"

Table 11 Categories defined through open coding for the commit messages in which a license change occurred.

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 FOR A LICENSE	
Licensing Request	0
LICENSE OUTPUT FOR THE END USER	
Output Licensing	0

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.

5 Lessons and Implications

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.

An important observation from our open coding analysis also stresses the need for better licensing traceability and aid in explaining the license grants/restrictions. We found several instances in which the issue tracker was used to ask for clarifications regarding licensing by external developers who sought to reuse the code. For example, we observed that developers interpret the implications of licensing differently, which generates misunderstandings in terms of reuse. This suggests that **code reuse is problematic for developers due to licensing. Therefore, our study demonstrates a need for clear and explicit licensing information for the projects hosted on a forge.**

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.**

Another important observation is the type of difficulty that developers have with the licensing of third-party code and the ways in which they achieve compliance. We observe in both the issue discussions and commit messages that libraries are removed due to incompatible licensing terms. Conversely, libraries are also chosen due to the particular license of the source code. This feature can be important for open source developers that aim for a wide adoption of their systems. Their choice in licensing may directly impact the adoption. This suggests that the choice in licensing can directly impact the adoption of libraries. Therefore, we foresee that **library/code recommenders based on open source code base should be license aware.** This consideration applies, for example, to approaches recommending code examples or libraries by sensing the developers' context [44,58,72,73]. In other words, on one hand the project's license should be a relevant part of the context, on the other hand, the code search engines (*e.g.,* [57,63–67]) should consider the target code license as a constraint in the search.

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* [68] and *ChangeScribe* [43,60]. While *ARENA* analyzes and documents li-

censing 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). **In summary, it would greatly improve traceability between license changes and their rationale, and assert a consistency among the repositories**. Also, it would be beneficial for developers using another project to be informed when a licensing change occurs. For example, a developer could mark specific projects as dependents and receive automated notifications when particular changes occur. This would be very beneficial with licensing since a change in the license of a dependency could result in license incompatibilities.

The open coding of commit messages and issue tracker discussions also suggests that commercial usage of code is a concern in the open source community. Currently, the *MIT/X* license and the *Apache* license seem to be the most prominent licenses for this purpose. Indeed, also the quantitative analysis of Java projects showed a trend towards the use of permissive licenses. The lack of a license is an important consideration in open source development, since it suggests that the code may in fact be closed source (or copyrighted by the original author). We observed such issues in discussions related to lack of licensing, since it hindered reuse. Indeed, sometimes developers initiate an open source project without attributing it a license. This is either because they lack a deep knowledge of the importance of the licensing on the possibility of (dis)allowing certain types of reuse for their code [78], but also because **there is limited support in the task of choosing the most suitable license for a project**. Existing tool support, such as *Choose A License*[11], helps users in choosing a license, but the tool is completely context-insensitive with respect to the constraints imposed. A better, context-sensitive tool support is provided in the Markos project [39], but it mainly provides the list of compatible licenses for a given component.

6 Threats to Validity

Threats to *construct validity* concern the relationship between theory and observation, and relate to possible measurement imprecision when extracting data used in this study. In mining the *Git* repositories, we relied on both the GitHub API and the `git` command line utility. These are both tools under active development and have a community supporting them. Additionally, the GitHub API is the primary interface to extract project information. We cannot exclude imprecision due to the implementation of such API. In terms of license classification, we rely on *Ninka*, a state-of-the-art approach that has been shown to have 95% precision [55]; however, it is not always capable of identifying the license (15% of the time in that study). For what concerns the open coding performed in the context of $\mathbf{RQ}_4$, we have identified, through a stratified sampling, a sample of commit messages and issue tracker discussions large enough to ensure an error of $\pm 10\%$ with a confidence level of 95%. Such a sample has been identified starting from candidate commit messages and discussions identified by means of pattern matching, using the keywords of Table 1. Although we

[11] `http://choosealicense.com`

aimed to build a comprehensive set of licensing-related keywords, it is possible that we missed licensing-related discussions not matching any of these keywords.

Threats to *internal validity* can be related to confounding factors, internal to our study, that could have affected the results. For the *atomic licensing changes*, we reduced the threat of having the project size as a confounding factor by representing the presences of a particular change at each commit. A license change typically is handled at a given instance and not frequency. By using commit-level analysis, we prevent the number of files from inflating the results so that they do not inappropriately suggest large numbers of changes occurred in a project. To analyze the changes across projects, we took a binary approach of analyzing the presence of a pattern. Therefore, a particular project would not dominate our results due to size. To limit the subjectiveness of the open coding, classifications were always performed by two of the authors, and then every case of discording classification was discussed as explained in Section 3.3.

Threats to *external validity* represent the ability to generalize the observations in our study. Our quantitative study is based on the analysis of over 16K Java projects. This makes us confident that our findings have a good generalizability for what concerns Java systems, while they cannot be extended to systems written in other programming languages. Our qualitative study has been performed instead on commit messages and issue discussions extracted from software systems written in seven different languages. However, the generalizability of our qualitative results is limited to the seven considered languages and it is supported by the relatively low number of considered systems (*i.e.,* 1,160) due to the manual effort required for the identification of the rationale behind licensing decisions (as well as the limited number of potential repositories with license-related commit messages or issue discussions).

GitHub's exponential growth and popularity as a public forge indicates that it represents a large portion of the open source community. While the exponential growth or relative youth of projects can be seen as impacting the data, these two characteristics represent the growth of open source development and should not be discounted. Additionally, GitHub contains a large number of repositories, but it may not necessarily be a comprehensive set of all open source projects or even all Java projects. However, the large number of projects in our dataset (and relatively high diversity metrics values as shown in Section 3.4) gives us enough confidence about the obtained findings. Further evaluation of projects across other open source repositories (and other programming languages for the quantitative part) would be necessary to validate our observations in a more general context. It is also important to note that our observations only consider open source projects. Since we need to extract licenses from source code, we did not consider any closed source projects and we cannot assert that any of our results would be representative in closed source projects.

7 Conclusions

This paper reported an empirical study aimed at analyzing, from a quantitative and qualitative point of view, the adoption and change of licenses in open source projects hosted on GitHub. The study consists of (i) a quantitative part, in which we stud-

ied license usage and licensing changes in a set of 16,221 Java projects hosted on GitHub, and (ii) a qualitative analysis in which we analyzed commit messages and issue tracker discussions from 1,160 projects hosted on GitHub and developed using seven most popular programming languages (*i.e.,* C, C++, C#, Java, Javascript, Python, and Ruby).

The quantitative analysis on the Java projects aimed at (i) providing an overview of the kinds of licenses being used over time by different projects, (ii) analyzing licensing changes, and (iii) identifying traceability links between licensing changes and licensing-related discussions. Results indicated that:

- New license versions were quickly adopted by developers. Additionally, new license versions of restrictive licenses (*e.g., GPL-3.0* vs *GPL-2.0*) favored longer survival of earlier versions, unlike the earlier version of permissive licenses that seem to disappear;
- Licensing changes are predominantly toward or between permissive licenses, which ease some kind of derivative work and redistribution, *e.g.,* within commercial products;
- There is a clear lack of traceability between discussions and related license changes.

The qualitative analysis was based on an open coding procedure inspired by grounded theory [42], and aimed at categorizing licensing-related discussions and commits. The results indicate that:

- Developers post questions to the issue tracker to ascertain the project's license and/or the implications of the license suggesting that licensing is difficult;
- There is a lack of standardization or consistency in how licensing is attributed to a system (both within the same programming language and across different programming languages), which causes misunderstandings or confusion for external developers looking to reuse a system;
- Developers, in general, do not supply detailed rationale nor document changes in the commit messages or issue tracker discussions;
- License compatibility can impact both the adoption and removal of a third-party library due to issues of license compliance.

This work is mainly exploratory in nature as it is aimed at empirically investigating license usage and licensing changes from both quantitative and qualitative points of view. Nevertheless, there are different possible uses one can make of the results of this paper. Our results indicate that developers frequently deal with licensing-related issues, highlighting the need for developing (semi)automatic recommendation systems aimed at supporting license compliance verification and management. Additionally, tools compatible or integrated within the forge to support licensing documentation, change notification, education (*i.e.,* picking the appropriate license), and compatibility would benefit developers attempting to reuse code. While working in this direction, one should be aware of possible factors that could influence the usage of specific licenses and the factors motivating licensing changes. This paper provides solid empirical results and analysis of such factors from real developers.

Future work in this area should aim at (i) extending the study by performing a larger quantitative and qualitative analysis on more projects, and (ii) performing a

deeper investigation into the rationale for licensing changes, for example, by performing an analysis of dependencies in software projects and relating such analysis with the changes being performed. Last, but not least, as discussed in Section 5, it would be useful to incorporate licensing analysis into existing software recommender systems. Such recommenders could not only rely on the local project's context, but also exploit rationale from previous licensing changes to produce recommendations.

Acknowledgements

This work is supported in part by NSF CAREER CCF-1253837 grant. Massimiliano Di Penta is partially supported by the Markos project, funded by the European Commission under Contract Number FP7-317743. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

References

1. 123done issue 139. https://github.com/mozilla/123done/issues/139.
2. android-sensorium issue 11. https://github.com/fmetzger/android-sensorium/issues/11.
3. brackets issue 8337. http://github.com/adobe/brackets/issues/8337.
4. The BSD 2-Clause License. http://opensource.org/licenses/BSD-2-Clause. Last accessed: 2015/01/15.
5. Cuanto commit. https://github.com/ttop/cuanto/commit/a1e58f2c93de40ab304c494e05853957c549fd44.
6. Czmq commit. https://github.com/zeromq/czmq/commit/eabe063c2588cde0af90e5ae951a2798b7c5f7e4.
7. d3-armory issue 5. https://github.com/kovmarci86/d3-armory/issues/5.
8. enigma2 commit. https://github.com/openatv/enigma2/commit/b4dfdf09842b3dcacb2a6215fc040f7ebbbb3c03.
9. F-Droid. https://f-droid.org/. Last accessed: 2015/01/15.
10. GitHub API. https://developer.github.com/v3/. Last accessed: 2015/01/15.
11. GNU General Public License. http://www.gnu.org/licenses/gpl.html. Last accessed: 2015/01/15.
12. gtksourcecompletion issue 1. https://github.com/chuchiperriman/gtksourcecompletion/issues/1.
13. gubg commit. https://github.com/gfannes/gubg.deprecated/commit/4d291ef433f0596dbd09d5733b25d27b3a921cf4.
14. Haml commit. https://github.com/haml/haml/commit/537497464612f1f5126a526e13e661698c86fd91.
15. Intex issue 1. https://github.com/mtr/intex/issues/1.
16. jackson-module-jsonschema issue 35. https://github.com/FasterXML/jackson-module-jsonSchema/issues/35.
17. jquery-browserify issue 20. https://github.com/jmars/jquery-browserify/issues/20.
18. minixwall commit. https://github.com/booster23/minixwall/commit/342171fa9e9d769ce4aa48525142a569b34962f7.
19. neunode issue 5. https://github.com/snakajima/neunode/issues/5.
20. Nimble commit. https://github.com/bradleybeddoes/nimble/commit/e1e273ff18730d2f8e0d7c2af1951970e676c8d1.
21. Passenger issue 1482. http://github.com/phusion/passenger/issues/1482.
22. patchelf issue 37. https://github.com/NixOS/patchelf/issues/37.

23. PF: The OpenBSD Packet Filter. http://www.openbsd.org/faq/pf. Last accessed: 2015/01/15.
24. Postgis commit. https://github.com/postgis/postgis/commit/4eb4127299382c971ea579c8596cc41cb1c089bc.
25. pyelection issue 1. https://github.com/alex/pyelection/issues/1.
26. python-hpilo issue 85. https://github.com/seveas/python-hpilo/issues/85.
27. rcswitch-pi issue 17. https://github.com/r10r/rcswitch-pi/issues/17.
28. Ros-comm commit. https://github.com/ros/ros_comm/commit/e451639226e9fe4eebc997962435cc454687567c.
29. schevorecipe.db commit. https://github.com/Schevo/schevorecipe.db/commit/b73bef14adeb7c87c002a908384253c8f686c625.
30. Software Package Data Exchange (SPDX). http://spdx.org. Llast accessed: 2015/01/15.
31. State of the Octoverse in 2012 https://octoverse.github.com/. Last accessed: 2015/01/15.
32. Steampp issue 1. https://github.com/seishun/SteamPP/issues/1.
33. svgeezy issue 20. https://github.com/benhowdle89/svgeezy/issues/20.
34. tablib issue 114. https://github.com/kennethreitz/tablib/issues/114.
35. Tardis commit. https://github.com/tardis-sn/tardis/commit/07b2a072d89d45c386d5f988f04435d76464750e.
36. web-workshops issue 1. https://github.com/rosedu/web-workshops/issues/1.
37. wkhtmltopdf-qt-batch commit. https://github.com/alexkoltun/wkhtmltopdf-qt-batch/commit/9b142a07a7576afa15ba458e97935aac5921ef8d.
38. G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. The evolution of project inter-dependencies in a software ecosystem: The case of apache. pages 280–289, 2013.
39. G. Bavota, A. Ciemniewska, I. Chulani, A. De Nigro, M. Di Penta, D. Galletti, R. Galoppini, T. F. Gordon, P. Kedziora, I. Lener, F. Torelli, R. Pratola, J. Pukacki, Y. Rebahi, and S. G. 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.
40. A. Brock. Project Harmony: Inbound transfer of rights in FOSS Projects. *Intl. Free and Open Source Software Law Review*, 2(2):139–150, 2010.
41. S. Cass. The 2015 top ten programming languages.http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages.
42. J. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.
43. L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. 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.
44. D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Software Eng.*, 31(6):446–465, 2005.
45. M. Di Penta, D. M. Germán, and G. 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.
46. M. Di Penta, D. M. Germán, Y. Guéhéneuc, and G. 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.
47. D. A. Dickey and W. A. Fuller. Distributions of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74:427431, 1979.
48. D. A. Dickey and W. A. Fuller. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica*, 49(4):10571072, 1981.
49. B. Doll. The octoverse in 2012 http://tinyurl.com/muyxkru. Last accessed: 2015/01/15.
50. R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: a language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 422–431, 2013.
51. Free Software Foundation. Categories of free and nonfree software. https://www.gnu.org/philosophy/categories.html. Last accessed: 2015/01/15.

52. D. M. Germán, M. Di Penta, and J. 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.
53. D. M. Germán, M. Di Penta, Y. Guéhéneuc, and G. 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.
54. D. M. Germán and A. 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.
55. D. M. Germán, Y. Manabe, and K. 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.
56. R. 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.
57. M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. A search engine for finding highly relevant applications. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 475–484, New York, NY, USA, 2010. ACM.
58. R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 117–125, 2005.
59. J. Howison, M. Conklin, and K. Crowston. FLOSSmole: a collaborative repository for FLOSS research data and analyses. *IJITWE'06*, 1:17–26.
60. M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk. ChangeScribe: A tool for automatically generating commit messages. In *37th IEEE/ACM International Conference on Software Engineering (ICSE'15), Formal Research Tool Demonstration*, page to appear, 2015.
61. Y. Manabe, Y. Hayase, and K. 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.
62. Y. Manabe, Y. Hayase, and K. 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, 2010.
63. C. McMillan, M. Grechanik, and D. 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.
64. C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie. Exemplar: A source code search engine for finding highly relevant applications. *Software Engineering, IEEE Transactions on*, 38(5):1069–1087, Sept 2012.
65. C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: Finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 111–120, New York, NY, USA, 2011. ACM.
66. C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher. Recommending source code for use in rapid software prototypes. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 848–858, Piscataway, NJ, USA, 2012. IEEE Press.
67. C. Mcmillan, D. Poshyvanyk, M. Grechanik, Q. Xie, and C. Fu. Portfolio: Searching for relevant functions and their usages in millions of lines of code. *ACM Trans. Softw. Eng. Methodol.*, 22(4):37:1–37:30, Oct. 2013.
68. L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. 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.
69. M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 466–476, 2013.
70. Oracle. MySQL - FOSS License Exception. `http://www.mysql.com/about/legal/licensing/foss-exception/`. Last accessed: 2015/01/15.

71. M. D. Penta and D. 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.
72. L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In *17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013*, pages 57–66, 2013.
73. L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the IDE into a self-confident programming prompter. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 102–111, 2014.
74. P. Singh and C. Phelps. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, 24(3):539–560, 2009.
75. M. Sojer and J. 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.
76. T. Tuunanen, J. Koskinen, and T. Kärkkäinen. Automated software license analysis. *Autom. Softw. Eng.*, 16(3-4):455–490, 2009.
77. C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. Germán, and D. 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*. IEEE, 2015.
78. C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. German, and D. 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.
79. Y. Wu, Y. Manabe, T. Kanda, D. M. Germán, and K. 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.
80. C. Zapponi. Githut.`http://githut.info`.