Open Source License Inconsistencies on GitHub

THOMAS WOLTER, Friedrich-Alexander University Erlangen-Nuernberg
ANN BARCOMB, Schulich School of Engineering, University of Calgary
DIRK RIEHLE, Friedrich-Alexander University Erlangen-Nürnberg
NIKOLAY HARUTYUNYAN, Friedrich-Alexander University Erlangen-Nuernberg

Almost all software, open or closed, builds on open source software and therefore needs to comply with the license obligations of the open source code. Not knowing which licenses to comply with poses a legal danger to anyone using open source software. This article investigates the extent of inconsistencies between licenses declared by an open source project at the top level of the repository, and the licenses found in the code. We analysed a sample of 1,000 open source GitHub repositories. We find that about half of the repositories did not fully declare all licenses found in the code. Of these, approximately ten percent represented a permissive vs. copyleft license mismatch. Furthermore, existing tools cannot fully identify licences. We conclude that users of open source code should not only look at the declared licenses of the open source code they intend to use, but rather examine the software to understand its actual licenses.

CCS Concepts: • **Software and its engineering** → *Open source model*; *Risk management*; • **Social and professional topics** → *Licensing*.

Additional Key Words and Phrases: license management, license conflicts

ACM Reference Format:
Thomas Wolter, Ann Barcomb, Dirk Riehle, and Nikolay Harutyunyan. 2022. Open Source License Inconsistencies on GitHub. 1, 1 (October 2022), 21 pages. https://doi.org/TBD

1 INTRODUCTION

Open source software is used in almost all of today's software products. As a result, open source software has become ingrained in the modern software engineering. It provides a large number of solutions to challenges that software developers might face, from the scope of small applications up to operating systems. A report by the European Commission estimates that the use of open source software saves the European economy about €114 billion per year directly in development costs. Furthermore, an additional value of about €342 billion is created by increasing productivity and efficiency through these savings [11]. Our work fits in with the body of research viewing open source software through the lens of consumer companies [1, 29, 37, 50]. Project licensing plays a critical role in the relationship between open source projects and companies, and licence compliance remains a concern of companies [9, 47]. In our approach we follow Fitzgerald, who proposed that open source software has transformed from a purely social movement to a more commercial endeavor [15].

Authors' addresses: Thomas Wolter, tho.wolter@fau.de, Friedrich-Alexander University Erlangen-Nuernberg; Ann Barcomb, ann@barcomb.org, Schulich School of Engineering, University of Calgary; Dirk Riehle, dirk@riehle.org, Friedrich-Alexander University Erlangen-Nürnberg; Nikolay Harutyunyan, nikolay.harutyunyan@fau.de, Friedrich-Alexander University Erlangen-Nuernberg.

However, the use of open source components in products must comply with the license the components have been published under. Lack of compliance to these terms can result in a variety of problems, ranging from easily resolved disagreements to lengthy legal disputes. In the worst case, court-ordered injunctions can lead to an immediate sales stop of the product.

Given the broad use of open source software from public repositories in products and the potential legal problems resulting from failures to comply with licenses, we wanted to expand upon previous research on the subject of license mismatches and inconsistencies [21]. Our goal was to find out how much code on GitHub, the preeminent open source code hosting service, had an incorrect or incomplete license declaration in the repository summary.

We identify two ways in which the license can be specified. A **declared license** is the license specified for the whole project, for instance in a LICENSE file in the project's root folder or displayed in the GitHub UI. By contrast, **in-code licenses** are the licenses attached directly to files along the entire directory tree. Both declared and in-code licenses are **explicit licenses** in that they are provided as part of the repository. This opens up the possibility that a repository's declared license differs from one or more in-code licenses. This could occur, for instance, if an open source repository includes an external software library and does not properly document the corresponding license as a declared license. As a result, a user might not be aware of the license terms and could accidentally ignore the in-code license obligations.

One recent example which demonstrates the complications which can arise from license inconsistency occurred in March, 2021.[1] The `mimemagic` software library, which was distributed under a (declared) MIT license, incorporated the `shared-mime-info` library, which is distributed under the GPL license (a more restrictive license than the MIT license). The `shared-mime-info` copyright notice was stripped by a merge tool, meaning that `mimemagic` users had to examine the library in the repository to find the (in-code) license. This license mismatch had a significant impact: the `mimemagic` library was required by the Ruby on Rails web framework and affected 172 other packages, impacting an estimated 577,000 software repositories, leading to an urgent effort to resolve the problem.[2]

Our observation of the potential for license conflict and its possible impact—a problem which is acknowledged by practitioners and which has spurred the development of license scanners to identify in-code licenses—leads to the following research questions:

- **RQ1**: How often does the declared license of an open source GitHub repository differ from the in-code licenses?
- **RQ2**: How often does the declared license have a different level of restrictiveness than in-code licenses?
- **RQ3**: How often is a permissive declared license contradicted by a more restrictive copyleft in-code license?

Our work has significant implications for practitioners, as it draws attention to a widespread lack of licensing clarity in open source software. We found that about ten percent of the repositories in our sample had a declared permissive license, while containing copyleft licensed code.

We also evaluated two of the most popular license scanners which are currently used in industry to identify and assess open source licenses. Researchers tracking the prevalence of different licenses can also benefit from clear evidence on the state of licensing in popular GitHub repositories. The contributions of this article are:

- An analysis and classification of how licenses are expressed in practice,
- An evaluation of accuracy of existing license scanners,
- A quantitative measure of insufficient license labeling of GitHub repositories, and
- A risk assessment of the insufficient labeling using different license types.

[1] https://perma.cc/98DC-MWAU, https://perma.cc/3XNU-8LHB
[2] https://perma.cc/PC2J-24QF

The article is structured as follows: we first present the background of our study in section 2, followed by a review of related work in section 3. Next, we describe our research method in section 4. There, we define different types of licenses, how we gathered data, and how we verified its accuracy. In section 5 we present our results, most notably the significance of insufficiently labeling repository licensing and the associated risks for open source users. In section 6 we discuss our findings, while in section 7 we discuss threats to validity. In section 8 we close the paper with our final conclusions and recommendations.

2 BACKGROUND

2.1 Motivation

Open source licensing has been chiefly studied as a topic of open source software research. The Open Source Initiative (OSI) provides a definition of open source software[3] which specifies criteria which must be fulfilled for a licence to be considered open source. Research, therefore, has often focused on the differences in restrictiveness between open source licenses [4][17][39][53]. A higher level of restrictiveness usually correlates with more obligations one must fulfill while using the open source code. In the strongest form, a copyleft clause can require users to make any changes to open source code publicly available under the same license it was originally published under [43]. An example of the copyleft effect can be seen in the GPL family of licenses (GNU General Public License[4]). The most recent version, GPL-3.0, stipulates that software code that includes or modifies code licensed under GPL-3.0 also needs to be published under the GPL-3.0 license. As a result, the code in a repository licensed under GPL-3.0 will always remain so and any derivatives must also adapt to this license. A permissive license on the other hand does not impose these restrictions and allows users to introduce open source code into their own repository, without having to disclose their own source code [3][16]. Consequently, from a business standpoint, the choice of which licenses are suitable for inclusion in one's own source code is of importance. More restrictive licenses are generally disliked by vendors, as having to publish their proprietary code under an open source license does not align with most business models. Permissively licensed code on the other hand can be used to create closed proprietary software. Thus, in most cases permissively licensed open source repositories are preferred by software vendors, while copyleft repositories are avoided [24][45][46][51]. Besides the commercial viability, it is also important to consider that not every license is compatible with every other license, as the obligations can contradict each other. Thus, the use of one license can automatically rule out the correct use of other licensed code [18][20].

With increasing use of open source software in products, vendors of these products need to ensure that only suitably licensed open source code is incorporated. Failing to comply with licensing terms, knowingly or unknowingly, can result in offenders being embroiled in lengthy legal disputes, having to pay monetary compensation, and receiving cease and desist orders that can stop a product's distribution entirely [7][28][48][51]. To prevent damaging effects, companies and open source communities can practice preventative measure with open source governance [18][27][38]. In the context of this paper, we want to primarily focus on license compliance role of open source governance. This can for example be done by ensuring licenses are interpreted correctly, any identified license is documented, and used open source code is approved for use [26]. These tasks are designed to avoid introducing any potential unwanted license terms and their obligations.

In order for these strategies to work, it is however important that licensing information is clearly marked and can easily be found. In a preliminary study, however, we found numerous cases of multiple, often hidden licenses in

[3]https://opensource.org/osd
[4]The GNU General Public License – https://www.gnu.org/licenses/licenses.html

repositories. In some cases, we found inconsistencies with GitHub repositories that claim to be permissively licensed on the top level, but contain GPL or other copyleft licensed code. As such, repositories exist that have copyleft source code hidden along the directory tree. Thus, it is possible that common open source governance methods are unable to find and track these licenses. This poses a threat to any commercial product which incorporates the code, as unwanted license terms may come with the incorporated code.

Recent studies demonstrate that open source license violations are not random accidents, but a large scale problem encompassing most software and hardware products with open source components in their binaries [14], despite the increasing number of tools and frameworks including findOSSLicense [35], OpenChain [8] and others, which can help with open source governance and license detection and compliance.

2.2 License Inconsistencies and Conflicts

We use the term *inconsistency* to refer to the use of two different licenses within the same project. Inconsistency does not necessarily imply *conflict*, which occurs when two licenses contain contradictory rights or incompatible obligations.

License inconsistencies can lead to particularly severe license conflicts if the undocumented in-code licenses are more restrictive than the declared licenses. We refer to inconsistencies with a different level of license restrictiveness as *mismatches*. Mismatches do not necessarily imply a conflict, but increase the likelihood that one exists.

In order to determine the number of such license conflicts in our sample, we classified open source licenses into four different categories based on their level of restrictiveness:

- Permissive: Changes to existing code can but do not have to be published under the same license.
- (Strong) copyleft license: Changes to existing code and all code using the existing code must be published under the same license.
- Weak copyleft license: Changes to existing code must be published under the same license; code using the existing code does not have to.
- Other: All licenses that do not fit into any of the above license categories.

The four categories were chosen to showcase conflict situations that could potentially lead to legal ramifications. A differentiation between permissive and copyleft licenses was necessary, because the introduction of copyleft-licensed code can cause unwanted consequences in a commercial setting. A further distinction between copyleft and weak copyleft was made, because of the difference in their effects. The 'Other' category was added because some licenses can not be classified into the existing categories. This, for example, includes licenses with public domain implications. The classification of each license was extracted from DejaCode's license list[5] (DejaCode license format is supported by ScanCode and owned by the same parent company).

2.3 License Expression

We identified two ways in which a software license can be expressed in a GitHub repository:

(1) A *declared license* is an open source software license that has been set (declared) by the repository owner. There are two main ways of doing so: By using the GitHub user interface to specify the license, and by manually providing a LICENSE or README file that contains or specifies the license in the root directory of the repository. If more than one license is declared, the declared license can be a set.
(2) An *in-code license* is the set of software licenses found inside the entire directory structure, either as stand-alone LICENSE files or within source code files (usually in the file header).

[5]https://enterprise.dejacode.com/licenses/

For a given repository, it is possible that declared and in-code licenses differ. An example for this is a repository that contains multiple licenses along the directory tree. In this case the declared license information will not contain the licenses added in subdirectories, unless mentioned on the top level. Furthermore, we also deal with differing declared licenses. This can for example occur, if a repository's LICENSE and README file name different licenses. This can potentially lead to the license declarations contradicting each other. Inconsistencies between different in-code files are not a part of this paper.

2.4 Automated Open Source Scanning

In today's software development a manual check by reading through each source code file is not a feasible option, as it would be too time consuming and potentially error prone [10][23][33][44]. Furthermore, researchers often deal with large samples of software repositories and are faced with the same problem [20][56][58]. In order to manage, identify, and prevent open source license inconsistencies, developers and researchers have started to support the license compliance aspect of open source governance with software tools such as license scanners. These tools aim to analyze license files or the whole code base of an open source repository. As a result, they can provide a digested view of the identified licenses and have the potential to find license notices hidden deep in the code base [12]. However, a developer (or a compliance officer in case of some companies) then needs to manually review and resolve the incompatible license combinations to ensure compliance. Despite creating extensive overhead for software development, these tools are nonetheless essential to open source license compliance [19].

There are a large number of license scanners (commercial and open source) available today. Some of the earliest academic research on potential solutions included the Automated Software License Analyzer (ASLA) and Ninka. ASLA was able to identify the license of about 89% of source code files in a sample of 12 open source packages [54][55]. Ninka achieved a recall of 82.3% and a precision of 96.6% in a sample of 0.8 million source code files in Debian 5.0.2 [22]. Overall, the early studies concluded that a reliable approach was feasible, but at times manual input was necessary. However, neither aproach is still supported today, as ASLA was last updated in 2015[6] and ninka was last updated in 2017[7]. For our study we decided to pick FOSSology[8] (an open source project of the Linux Foundation) and ScanCode[9] (an open source project of nexB), two of the most popular open source license scanners available today. These tools were chosen because of their widespread use and acceptance by both the industry [13] and academia [59], in addition to their affordability (both available as open source software). Furthermore in our preliminary study we determined these products substantially outperformed other alternatives [57].

In order to find licenses, the scanners deploy different approaches. Nomos scans for licenses by using keywords to search for license relevant statements in the codebase, then employing regular expressions to determine the actual license from this section[10]. ScanCode uses a search index (a collection of license texts and license detection rules) that is queried with extracted text from files[11]. For a more detailed explanation of the algorithms used by each scanner, we refer to the software documentation. We also calculated the hybrid result of both scanners in order to identify and analyze license declarations in our sample. In this work, we analyze and discuss some of the limitations of the license

[6] ASLA – https://sourceforge.net/projects/asla/
[7] Ninka – https://github.com/dmgerman/ninka
[8] The FOSSology project – https://www.fossology.org
[9] ScanCode – https://github.com/nexB/scancode-toolkit
[10] Licenses with Nomos – https://www.fossology.org/features/
[11] Licenses with ScanCode – https://scancode-toolkit.readthedocs.io/en/latest/explanations/overview.html#how-does-scancode-detect-licenses

scanning algorithms. While out of our scope, we see better tooling as a potential long-term solution to open source license confusion.

3 RELATED WORK

In the previous section, we discussed our preliminary work on open source licensing and the related work directly tied to the context of this study, in order to present the reader with the background necessary to understand our approach. We continue with a broader discussion of the state-of-the-art literature in this section, highlighting the topics of the research evolution on open source licensing, as well as existing publications identifying or quantifying license mismatches and inconsistencies found in practice.

3.1 Evolution of Open Source Licensing

Open source licensing and the identification of the potential license and copyright violations has been a topic of both industry and academic research for years. Hemel et al. developed the Binary Analysis Tool (BAT), a system for code clone detection in binaries to assess the scale of the problem in 2011 [31]. They evaluated different clone detection techniques and eventually used the method of scanning for string literals. At the time, this was an efficient way of finding previously unidentified open source code in software and hardware products (such as products with embedded Linux software and other open source packages).

The same authors revisited the subject ten years later to reassess their work and its impact [32]. In this retrospective, they found that both industry and academia have evolved on the subject of license compliance and detection. Specifically, recent tooling, such as FOSSology [23] and findOSSLicense [35], as well as open source compliance initiatives, such as OpenChain [5, 8] and SPDX [25, 36], have made it easier for well-intentioned actors and companies to identify and comply with the open source licenses, by significantly decreasing their risks of accidentally violating licenses. However, we agree with Hemel et al. [32] that their scanning techniques and other methods in this area continue to remain prominent for catching malicious actors and for broad analysis of code containing open source software.

3.2 License Mismatches and Inconsistencies

Some of the principal investigators of license mismatches and inconsistencies, German and Hassan, acknowledge the wide-spread mismatches of the declared and actual licenses in open source repositories. They discuss different approaches to the integration of multiple licenses taking into account the existing mismatches in component-based software development [21]. Another study by German et al. analyzes the re-distribution of packages and attempts to create a semi-automatic license auditing method. The purpose is to find incompatibilities between licenses declared for a package (the license the package is distributed under) and other licenses found in the source code (referred to as 'source licenses') or the requirements of the package. While investigating the Linux-based Fedora 12 operating system, the authors found a variety of inconsistencies, including permissive–copyleft conflicts [20]. While this study shares some similarities with our approach, it is based on packages distributed with RedHat Fedora 12, while we specifically focus on repositories hosted on GitHub. Furthermore, as the study was conducted more than a decade ago, it may no longer accurately describe current licensing consistency. We provide a comparison with this study in the Result section.

Moraes et al. [42] describe that about 62% of repositories in their sample of 1,552 JavaScript projects showed the use of multiple open source licenses. However, only 30% actually have multiple licenses declared at the top-level. The other licenses are only declared at the file-level and can only be found by searching the entire code base. They go on to mention that a lot of developers lack the understanding of the relationships between the licenses and the consequences

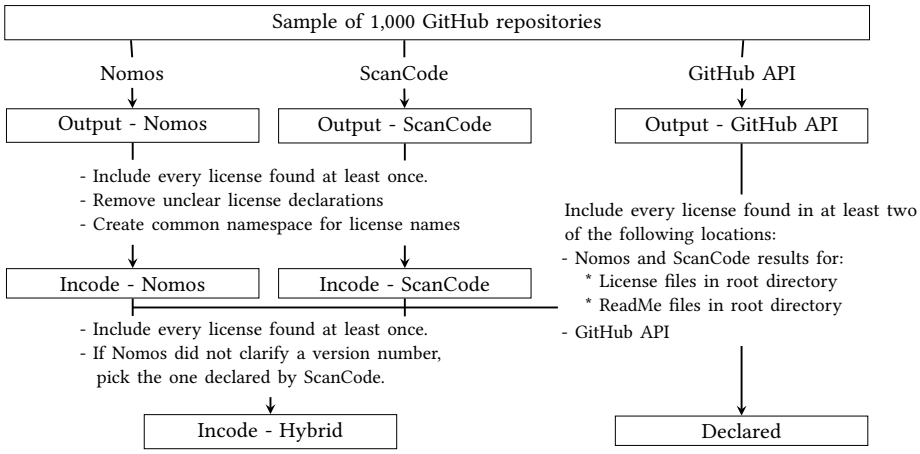

Fig. 1. Data parsing workflow

of using multiple licenses. The lack of understanding was also investigated by Almeidaet al. [2], as they found that software developers were mostly confident in cases with only one license, but showed weaknesses if multiple licenses were declared. While we do not propose a license integration matrix or compliance tool in this study, we do provide a method for discovering the various license inconsistencies when analyzing a database of GitHub repositories (see subsection 4.1).

Looking at recent studies about code duplication, license inconsistencies could prove to be a bigger problem than one might initially think. A study from Lopes et al. [40] showed that from 428 million files hosted on GitHub, only 85 million were unique. The remaining files were for example copied from larger repositories or are new forks of abandoned repositories. Furthermore, while studying provenance of software, Rousseau et al. [49] found that the replication factor of the raw byte sequence of files is high. The prevalence of software copying has also been observed in other contexts, for example from Stack Overflow to GitHub, although in this case license information may not be transferred [6]. Taking these studies into account, it is important to consider that license inconsistencies can also be duplicated and spread further.

4 RESEARCH METHOD

This section is split into three parts. Firstly, we give an overview of the terminology used for our quantitative analysis. This includes a classification of ways licenses can be declared in a open source repository. Secondly, we describe the choices we made for the data collection, as well as the tools used to analyse the licensing data. Lastly, we describe the measures we took to verify the validity of the analysis tools we used for license detection.

4.1 Data Collection

Our sample consisted of 1,000 randomly selected open source repositories from the intersection of GitHub and OpenHub[12]. GitHub is by far the largest open source forge [52], hosting over 100 million public repositories. GitHub therefore offers access to the largest collection of repositories through a single API. However, many of the repositories

[12]OpenHub – https://www.openhub.net/

are duplicates, inactive, largely untouched, for private use, or do not contain software [34]. OpenHub is a platform which seeks to include only genuine open source repositories, indexed from a variety of sources. Thus, we chose to only include GitHub repositories also cataloged by OpenHub. This was the only criteria we imposed on the repository selection and it allowed us to filter out inappropriate repositories and ensure that the chosen repositories had seen use by other developers. Overall, the average GitHub star rating—a metric which can indicate interest in the repository—was 289.64 stars and 75.4 percent of repositories had been forked at least once. Data collection was performed in April 2020.

As there are a total of 395,995 files in our sample, manually checking for licensing data was not feasible. Thus, we decided to extract the information in several different ways. For data found in the repository files, we used the scanning algorithms from FOSSology's Nomos tool and ScanCode. These scanners were chosen because our initial research showed that they performed better than other alternatives available [57]. We collected independent results from each scanner, in addition to developing a hybrid version for in-code licenses that took both scanners' output under consideration. Figure 1 gives an overview of the approach we took to construct the different license lists.

We used a Python script to create lists (Nomos, ScanCode, and hybrid) of declared and in-code licenses. For the declared licenses, we also used the GitHub API to collect metadata, and parsed the Nomos and ScanCode outputs for LICENSE and README files in the root directory. If a license was mentioned in multiple components, thus having either a consensus between Nomos and ScanCode or license hits in multiple locations, we added the specific instance to the result list. For the in-code results, we looked at each license scanner's output. If a license was identified anywhere along the entire directory tree, we added it to the list. For the hybrid solution, we checked for cases in which both scanners came to the same conclusion. Furthermore, we noticed that Nomos sometimes did not specify a version number for licenses. In this case, the hybrid version uses the parts of ScanCode's evaluation, if present, that share the same license family.

One of the issues we faced was finding a common namespace for license names between the scanners. We standardized on SPDX[13] license tags as identifiers. SPDX provides a list of commonly used licenses and is used by GitHub, Nomos and ScanCode. However, both scanners contained a variety of licenses not included in SPDX and thus had different descriptors. In this case, we created our own version of these tags similar to those in SPDX.

4.2 Data Verification

In order to ensure that the license scanners' output actually represented the ground truth, we decided to perform additional research on the scanners' ability to identify the declared and in-code licenses. We randomly picked a fixed sample of 250 repositories and 500 files from our initial sample and manually verified their contents. The repositories' contents were used to evaluate the declared licenses and the files were used to evaluate the in-code licenses. By comparing the scanners' evaluation with our manual check, we were then able to calculate the *margin of error*, *precision*, *recall* and *F-measure*. Table 1 shows the formulas used for each of these values.

In the context of this paper the values represent the following:

- The margin of error is used to show how likely a repeat of the sampling and evaluation would come to the same conclusion.
- The precision value describes the ratio of returned licenses to those actually present in the checked files or repositories. Thus it is an indication of the scanner's ability to mark the correct license.

[13]The Software Package Data Exchange – https://spdx.org/licenses/

Table 1. Formulas used for the verification process

Name	Formula
Margin of Error	$\text{z-score} * \sqrt{\frac{p * (1-p)}{n}}$
Precision	$\frac{\text{Correctly identified licenses}}{\text{Correctly identified licenses + Falsely identified licenses}}$
Recall	$\frac{\text{Correctly identified licenses}}{\text{Correctly identified licenses + Missed licenses}}$
F-Measure	$2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision + Recall}}$

- The recall value describes the ratio of retrieved licenses to the total amount present in the checked files or repositories. Thus, it is a measure of the scanner's overall completeness.
- The F-measure describes the harmonic mean between precision and recall.

To calculate the margin of error, the file or repository—depending on the unit being evaluated—was regarded as a unit, and the scanner's evaluation needed to be completely correct. When calculating the precision, recall, and F-measure, each license was treated individually. Thus, if a repository or a file contained more than one license, the scanner was evaluated on each license that was present, and each license it found. The manual check was done by the first author. Each file or repository was individually investigated for potential licensing information, and this data was used as a basis for our calculations. Because licenses are either objectively present or not, and can be accurately identified manually, we did not consider it necessary for a second person to review the identifications.

Results were added to a confusion matrix and could be either a true positive, false positive (precision error), true negative, or false negative (recall error). Imprecise findings, such as faulty version numbers, were treated as false positives. An example of an imprecise declaration would be a license marked as *GPL-2.0-only* which was identified as *GPL-2.0-or-later* in manual evaluation. Finally, *unknown* and other tags that do not represent a specific license were not considered a valid evaluation and were ignored.

5 RESULTS

This section is split into two parts. Firstly, we give an overview of the results of the data verification process. Secondly, we detail the license inconsistencies between declared licenses and between declared and in-code licenses. Furthermore, we give insight into the severity of the identified inconsistencies.

5.1 Data Verification

A complete overview of our calculations for the margin of error can be seen in Table 2. Overall, we found that a repeat of our verification process would be within a margin of 2% to 3% for both in-code and declared. Across the three in-code variations, the number of correct evaluations is very similar among the different approaches. However, this only means that the software tools had a similar number of correct evaluations and not that the evaluations agreed in a overwhelming number of cases. Oftentimes, one software tool made a correct evaluation of a repository, while another made mistakes.

Table 2. Margin of error for the 95% confidence interval

	Declared	Nomos	ScanCode	Hybrid
No. files checked	250	500	500	500
No. correct evaluations	231	461	459	462
Margin of error	0.0328	0.0235	0.0240	0.0232

Table 3. Recall, Precision, and F-Measure for license scanners in-code and declared evaluation

	Recall	Precision	F-Measure
Declared	0.8977	0.9753	0.9349
In-code Nomos	0.7688	0.8255	0.7961
In-code ScanCode	0.7875	0.7456	0.7660
In-code Hybrid	0.7468	0.8613	0.8000

Table 3 shows the results for precision, recall and F-measure for both in-code and declared licenses. Overall, this confirmed our previous findings about ScanCode and Nomos. ScanCode performs better in regard to recall and worse in regard to precision. This is most likely because it appears to be more liberal in marking a license. Even small text passages are used as indicators for the presence as a license, even if they are potentially insufficient to make accurate identifications. Thus, ScanCode has more false positives, but compensates by finding more licenses. The hybrid approach strengthens the precision value, because of the requirement for license hits to be present in both software tools and the combination of Nomos and ScanCode results in uncertain cases. The recall value on the other hand takes a small hit, because of potential disagreements between both software tools. Furthermore, the results for declared were stronger than the in-code evaluations.

While checking the files manually we observed the following two factors that might have influenced the overall outcome:

- License identification in LICENSE files seems to be more accurate than in software code, as the files are dedicated to this purpose. This includes being more specific with license identifiers and adding the entire license text, thus making it easier to identify.
- Overall, there are fewer found license identifications in the sections we search for declared licenses compared to the entire code base.

5.2 License Inconsistencies

We considered two types of license inconsistencies within a repository: between different declared licenses and between in-code and declared licenses. Inconsistencies between different in-code licenses were not considered. These cases were excluded because we wanted to focus the paper on inconsistencies between licenses declared at the top level and those contained deeper in the repository, which are more likely to result in license compliance violations. A focus on inconsistencies between in-code licenses would exceed the scope of this paper, but is a topic we consider for future research. Figure 2 gives an overview of the steps.

5.2.1 Declared License Inconsistencies. Inconsistencies between declared licenses can occur when more than one license is declared for a repository, for instance one in the LICENSE file and the other in a README file. If these licenses do not

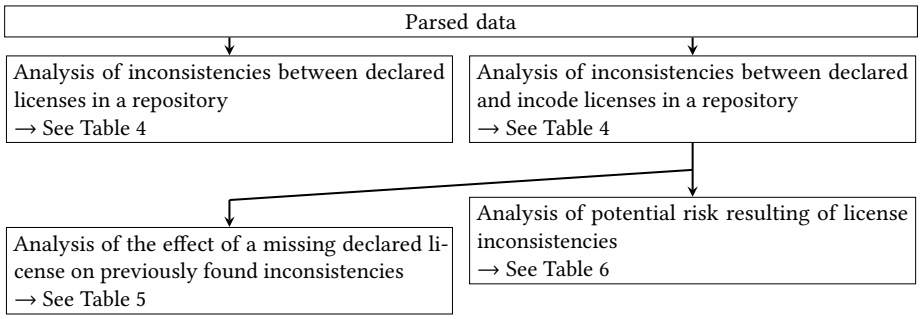

Fig. 2. Overview of the analysis process

Table 4. Total declared - declared and declared - incode license inconsistencies

	Consistent	**Inconsistent**
Declared - Declared	944	56
Declared - In-code Nomos	564	436
Declared - In-code ScanCode	451	549
Declared - In-code Hybrid	580	420

match, we consider this an inconsistency. As there are no definitive guidelines on where licenses are declared, we check specifically for cases in which there are multiple license findings in the three previously mentioned locations (GitHub API, LICENSE file, and README file). We ignore cases in which one location has a declaration and the others do not. For this we combined Nomos and ScanCode output for both LICENSE and README files. If there were different license findings among the designated locations, we marked it as an inconsistency. Table 4 shows an overview of the number of repositories in our sample displaying consistencies and inconsistencies between declared licenses as identified by our algorithm.

We observe that about five percent of the repositories from our sample showed inconsistencies between the different license locations. Furthermore, upon further investigation, we found that there were a total of 188 conflicts among the inconsistent repositories. Thus, repositories with faulty license declarations oftentimes have several such violations.

Observation 1

About five percent of repositories in our sample of 1,000 repositories showed one or more cases of inconsistency between declared licenses. Multiple inconsistencies were common, with an average ratio of 3.357 between inconsistencies and inconsistent repositories.

↗ Table 4

5.2.2 Declared and In-code License Inconsistencies. Inconsistency between declared and in-code licenses occurs when licenses are hidden in the code base and not declared on the top level. It is for example possible for a repository's license to be declared in a LICENSE file in the root directory and at the same time an undocumented license statement to exist in the header of a separate code file lower in the directory tree. If the licensing data between the two differs, we consider this as an inconsistency. An overview of all declared and in-code inconsistencies identified by Nomos, ScanCode, and a hybrid solution can be seen in Table 4.

Table 5. Disambiguated inconsistencies between found in-code and declared licenses

	License declared		No license declared	
	Correct	**Inconsistent**	**Correct**	**Inconsistent**
Nomos	354	219	210	217
ScanCode	266	307	185	242
hybrid	355	218	225	202

In regards to RQ1, we observe that even with the hybrid solution, which finds the least license inconsistencies, almost half of all repositories contain inconsistencies between in-code and declared licenses.

Observation 2

Approximately half of repositories have inconsistencies between the declared and in-code licenses.

↗ Table 4

The difference we observe between ScanCode and Nomos is likely due to the fact that ScanCode is more liberal when assigning licenses with unprovided version numbers. Furthermore, we noticed that a large number of repositories contain no license. In order to observe if this has an impact on inconsistencies, we split our original findings in separate cases, depending on whether a license was declared or not. Table 5 gives an overview of the disambiguated distribution.

Looking at the difference between cases with a declared license and cases without, we observe that the ratio of matches and inconsistencies is higher for cases with a declared license. Thus, if a license is declared in the first place, repositories seem to have stronger license compliance. The gap between ScanCode and the other solutions can again be explained by ScanCode's more liberal approach to marking license hits.

5.2.3 License Conflicts. As described in subsection 2.2, license inconsistencies can sometimes result in license conflicts. Although it is not possible to make a claim of conflict without examining each license pair, it is more likely that licenses in different categories will not only be inconsistent but in conflict.

Table 6 shows a complete overview of every found declared license in the entire sample sorted by type, matched to every corresponding in-code license sorted by type. This gives an overview of potential mismatches and conflict situations. The matrix contains the number for Nomos, Scancode, and the hybrid solution in each cell.

In regards to RQ2, we observe that there is a significant number of inconsistencies found by each license scanner. In regards to RQ3 it is especially important to consider that there is a large portion of cases in which a permissive license is declared, but weak copyleft or copyleft code can be found hidden in subdirectories. These cases are highlighted in the matrix with a grey background.

Observation 3

There is a significant number of in-code licenses with a different level of restrictiveness than the declared license.

↗ Table 6

Table 6. Relation of each found declared license to each found in-code license
(Values from top to bottom: Nomos, ScanCode, hybrid)

Declared license \ Other licenses in repository	Permissive	Weak Copyleft	Copyleft	Other
Permissive	537 896 551	139 299 156	203 343 255	32 311 28
Weak Copyleft	167 352 193	100 209 105	93 147 104	5 142 4
Copyleft	298 617 375	160 352 166	163 310 201	18 211 18
Other	39 69 44	13 27 14	6 11 8	1 29 2

Observation 4

For the declared permissive licenses, almost half of the found in-code licenses showed a higher level of restrictiveness.

↗ Table 6

Overall, these observations are consistent with the findings of German et al., who analyzed source code packages in Fedora 12. They found that many packages included source code licensed differently that the declared license [20].

We also noticed a small difference between ScanCode and the other two options. ScanCode continuously found more cases. In the case of the other category, this is because the scanner uses a larger database of licenses. Also, as we described, ScanCode is more liberal with its license hits, thus there are more cases for the other combinations.

6 DISCUSSION

Our research showed that the mishandling of license declarations is common in public GitHub repositories. Furthermore, we discovered that there are situations in which a copyleft license can be hidden in a permissively licensed repository. In the following subsections, we give our interpretation of the results in terms of the potential impact.

6.1 Potential Dangers of License Conflicts

As described in section 2, using repositories with copyleft licenses is generally avoided by most companies. To do so, companies practice open source governance. Observation 2 however shows that avoiding copyleft licenses is not as easy as one might think, because unclear licensing with hidden licenses is common. This is especially relevant in the case of permissive to copyleft inconsistencies. As mentioned in observation 4, we found that there was a significant number of cases that represent such situations. This poses a risk to any software repository, which may unknowingly

introduce unwanted license terms by importing a repository with possibly unclear or incomplete license declarations. This can have unintended implications for one's own software repository, including lengthy legal disputes. As our research shows, license scanners like Nomos and ScanCode can help making the license compliance process easier, however, the risk can not be eliminated completely.

It is however important to note that it is difficult to give a clear picture of what the exact consequences of such a violation are, because different factors can influence the outcome. One scenario is an amicable solution between both the license holder and the software user. Introducing code licenses under a GPL-3.0 license, for example, affords a time frame for the removal of such code, as the license provides the option to *"cure the violation prior to 30 days after your receipt of the notice"*[14]. This might however not be an easy option if a software repository is deeply ingrained in one's own repository. On the other side, if this option fails to resolve amicably, lawsuits are often filed. For this there is little data publicly available, as many cases are settled without going to court. However, this form of litigation usually involves monetary compensation.

The actual consequences can have different impacts on a software repository and are therefore difficult to quantify. However, we argue that even if any problems can be resolved amicably, conflicts should be avoided.

6.2 Relevance of License Conflicts

Our overall findings are based on our sample, however, we argue that this topic is relevant to the largest and most popular repositories that see a lot of commercial use. To confirm this, we decided to do a small scale repeat of our investigation with some of the most used repositories on GitHub, to see if there are cases with potential conflicts. For this, we picked the five repositories from GitHub with the highest star count. We recognize that the GitHub star rating is an imperfect metric for determining project popularity [41]. However, we wanted to use a criterion which could not be said to have been selected to prove our point (e.g., theoretical interest). We therefore used star count to ensure the example projects we examined had a certain measure of visibility and therefore potential impact.

We excluded repositories that do not actively develop code, such as book collections, because we wanted to focus on repositories that get used in commercial repositories. This left us with the following repositories:

- vuejs/vue : Vue
- twbs/bootstrap : Bootstrap
- facebook/react : React
- tensorflow/tensorflow : Tensorflow
- ohmyzsh/ohmyzsh : Ohmyzsh

Table 7. Conflict matrix for the five repositories (Values from top to bottom: Nomos, ScanCode, hybrid)

		Other licenses in repository			
		Permissive	Weak Copyleft	Copyleft	Other
Declared	Permissive	34 51 30	11 15 10	18 33 15	5 18 2

[14] GPL-3.0 license text – https://www.gnu.org/licenses/gpl-3.0.de.html

Table 8. Examples from top starred repositories

Case 1:

Name: Ohmyzsh	**Filepath:** plugins/gitfast/git-prompt.shl
# Copyright (C) 2006,2007 Shawn O. Pearce <spearce@spearce.org> # Distributed under the GNU General Public License, version 2.0.	

Case 2:

Name: Tensorflow	**Filepath:** tensorflow/tensorflow.bzl
# Config setting selector used when building for products # which requires restricted licenses to be avoided. def if_not_mobile_or_arm_or_lgpl_restricted(a):	

Case 3:

Name: Tensorflow	**Filepath:** third_party/eigen3/BUILD
# Note: Eigen is an MPL2 library that includes GPL v3 and LGPL v2.1+ code. # We've taken special care to not reference any restricted code.	

Table 7 shows the permissive row of the conflict matrix. The Permissive to Copyleft cell shows that there are also numerous cases in which a permissive license is paired with a more restrictive one. However, as we described earlier, this does not necessarily mean that the conflict is actually a risky situation. Thus, in this case, we decided to look at individual cases, to see if problems are likely to occur. Overall, a lot of the cases were false positives. Some of the files we checked were dual licensed, which does not pose the risks of a straight mismatch. Other files mentioned former licenses that are no longer valid. However, we did find some cases that pose a potential risk or show preventative measures for license mismatches. Table 8 gives an overview of some of the cases we deemed especially interesting.

In case 1, a *GPL* family license is found in one of the subdirectories of the repository. The declared license in the top level directory is however a permissive *MIT* license. The only indication for this is the advice in the README file to read each plugins' README file individually. It is however not explicitly pointed out that there might be a different license. This could cause a variety of problems, as mentioned before, for any user potential working with the code. Case 2 and 3 didn't directly show a potential conflict, however, they gave an impression that popular open source repositories pay close attention to licensing.

Case 2 was picked up by our algorithm, because a function contained the expression *lgpl*, even if it does not refer to an actual license declaration. However, we decided to include it, because the function implies that the creators actively tried to implement a way to filter out undesirable license types. This means that the creators of the repository are aware that potential license conflict might cause issues to future users and attempt to offer preventative measures. It also demonstrates why manual review is required when using license scanners.

Observation 5

The most starred repositories show evidence that some developers are aware of, and take effort to avoid license mismatch.

↗ Table 8

Case 3 shows that the developers working on the repository decided to include a new library that contains the weak

copyleft *MPL-2.0* license. However, some of the code is licensed under a stronger copyleft license. Thus, the developers were careful to only use and reference code that is published under the less restrictive license.

Thus, looking at some of the cases, we observed that even in some of the most used GitHub repositories the topic of licensing is relevant. We found cases in which copyleft licenses were not declared in permissive repositories. Furthermore, the cases in which developers take preventative measures to ensure license terms are complied to, showcase that popular repositories are aware of this topic.

6.3 Potential Correlations

To find if there is another way of potentially identifying repositories more likely to have license mismatches, we analysed the GitHub metadata corresponding to each repository in our data set. These metadata tags provide information on a variety of metrics of a repository. For this, we considered *size* (size of the repository in kB), *stargazers_count* (number of stars), *subscribers_count* (number of repository watchers), *forks_count* (number of forks from this project), *open_issues_count* (number of issues), *fork* (is the project a fork), *has_issues* (issues feature enabled), *has_projects* (projects feature enabled), *has_downloads* (downloads feature enabled), *has_wiki* (wiki feature enabled), and *has_pages* (a GitHub Pages site exists).

Table 9 gives an overview of the average value of each measure for all the projects in our sample, using the same categories provided in Table 5. The table shows, respectively, repositories where declared and in-code licenses are consistent (licenses declared - Correct), repositories where declared and in-code licenses are inconsistent (licenses declared - Inconsistent), repositories which contain neither declared nor in-code licenses and are thus internally consistent (No license declared - Correct), and repositories with no documented declared license with one or more in-code licenses are present (No license declared - Inconsistent). For each measure we show the results for Nomos, ScanCode, and our hybrid solution. For example, for projects with a declared license and matching in-code licenses, according to Nomos, the average size was 4057.84 kB.

Our purpose in examining the different measures provided by GitHub was to identify if there is a measure which is highly correlated with license inconsistency, which could potentially be used as proxy in order to indicate to users that a more detailed, and possibly manual, investigation of the licenses is advisable. Because the data were not normally distributed, we used the Kruskal-Wallis test to examine between group variance. While we performed the tests for all three approaches (Nomos, ScanCode, and hybrid), we here report only the hybrid results, as this method had slightly better performance. For more detailed results, refer to Appendix A.

The fork tag was not statistically significant at 99% ($p = .0144$), most likely because only a small number of repositories in our sample were forked from other repositories. While we did find some statistically significant variation between the groups for has_issues ($p = .0863$), has_projects ($p = .0094$), has_downloads ($p = .7076$), has_wiki ($p = .0003$), and has_pages tags ($p = .0021$), these are all boolean tags with less variability than the discrete variables. We found a statistically significant difference in repository size between the four groups ($p = .0000$). Larger repositories in our sample contain more conflicts than smaller ones. Likewise, subscribers_count, open_issues_count, forks_count, and stargazers_count were significant ($p = .0000$). In all cases, higher values were associated with inconsistencies. Further interpretation of these observations is beyond the scope of this study, but the relationships between repository size, number of subscribers, number of open issues, number of forks, and stargazers, and the likelihood of licensing inconsistencies presents an opportunity for future work.

Table 9. GitHub repository tags and their relation to the declared - in-code analysis

		License declared		No license declared	
		Correct	Inconsis-tent	Correct	Inconsis-tent
size [kB]	Nomos	4057.84	26490.76	4208.65	7474.50
	ScanCode	1627.58	22137.82	2588.27	8375.88
	hybrid	3936.21	26794.53	4196.99	7730.01
stargazers_count	Nomos	385.06	621.71	28.23	54.88
	ScanCode	271.73	652.28	20.79	57.81
	hybrid	388.04	617.92	28.44	56.62
subscribers_count	Nomos	18.13	41.16	4.95	7.76
	ScanCode	16.39	36.03	3.99	8.20
	hybrid	18.50	40.65	4.97	7.95
forks_count	Nomos	56.62	138.76	9.35	19.04
	ScanCode	49.36	121.40	7.08	19.78
	hybrid	56.67	139.06	9.20	19.96
open_issues_count	Nomos	9.43	38.34	0.88	4.90
	ScanCode	8.87	30.48	0.69	4.63
	hybrid	9.43	38.45	0.92	5.16
fork [%]	Nomos	1.13	4.15	0.48	1.38
	ScanCode	1.88	2.62	0.54	1.24
	hybrid	1.13	4.17	0.44	1.49
has_issues [%]	Nomos	96.89	93.09	95.24	95.39
	ScanCode	96.99	94.10	95.14	95.55
	hybrid	96.90	93.06	95.11	95.55
has_projects [%]	Nomos	94.63	95.39	99.05	98.16
	ScanCode	95.49	94.43	98.92	98.38
	hybrid	94.37	95.83	99.11	98.02
has_downloads [%]	Nomos	96.61	98.16	98.10	97.70
	ScanCode	96.61	98.16	98.10	97.70
	hybrid	97.18	97.22	97.78	98.02
has_wiki [%]	Nomos	82.20	82.95	93.33	89.86
	ScanCode	83.46	81.64	93.51	90.08
	hybrid	82.82	81.94	92.89	90.10
has_pages [%]	Nomos	13.56	9.68	3.81	12.90
	ScanCode	10.90	13.11	4.32	11.57
	hybrid	13.24	18.97	4.00	13.37

Observation 6

There is a positive correlation between project size and license mismatch. More stars and more subscribers are correlated with use of a declared license. Among repositories with a declared license, these two tags are positively associated with license inconsistencies.

7 LIMITATIONS

Following Heale and Twycross [30] to assess the rigor of our research through the criteria of validity and reliability, we identified the following key limitations.

To ensure the construct validity of our findings, we constantly had the sampling size in mind. The biggest challenge we faced while conducting our research was the amount of repositories and files that needed to be checked to create representative results. A manual check of each file would be too time consuming and possibly error prone. Thus, we used the license scanners to perform the evaluation process. While we were able to verify that the software tools produce relevant data, there is still room for potential errors we can not rule out completely. For example, that the crawlers cannot identify the context behind a statement. As such, cases that for example mention that a license is specifically avoided, or operating under a dual license, might be evaluated incorrectly, because the scanner only notes licenses that were found. We tried to gain a better understanding of this through our manual evaluation. Furthermore, it is important to mention that our choice of 500 random files for the verification of the declared license introduced a slight bias towards larger projects. As larger projects have more files to choose from, more files from such projects were picked. However, our sample still contained files from 151 different projects.

An additional concern is that GitHub contains projects which are not open source. We attempted to mitigate this problem by using only projects listed on OpenHub, as described in Section 4.1. We manually investigated the cases where the license was marked as 'other' as depicted on Table 6, indicating a possible non-open-source licence and determined that in all cases, either an open source license was present, or an open source license was misidentified by the scanner. Table 5 also depicts cases where no license was found, either declared or incode. In such cases we cannot confirm that the projects in question are open source. One possible explanation for their inclusion on OpenHub is that the GitHub repository represents a 'code dump,' with the main development of the project occurring on a different site. Our findings highlight the need for developers to clearly state license terms, regardless of which type of licensing is desired.

The next challenge was related to reliability, the degree of consistency of the findings and traceability from the data to the results. The limitation here was that some licenses could not clearly be identified by the license scanners. These situations exist because the scanners found some evidence for a license in a file, but are not able to recognize the exact license that is represented. Potential causes for this are messy license declarations (e.g. missing version numbers, spelling errors) or altered licensing text. As our scanning was completely automated due to the sample size, we were not able to determine the actual results for these cases and disregarded them in the final evaluation.

We tried to automate the license scanning and analysis as much as possible both for efficiency reasons and to reduce researcher bias, thus establishing a higher level of objectivity. The latter is the degree to which the authors are neutral towards the inquiry and their potential bias effect on the findings.

Lastly, we had to consider the external validity of our findings. One limitation that is important to mention is that the license scanners we used did not recognize code that was copied from other repositories or the web without a license declaration. Thus if code is copied and pasted from an online help forum without attribution, the license scanners used in our study would not be able to identify the violation. Our work only considers licenses which the authors have included in repositories, not all licenses which may legally be associated with a segment of code.

8 CONCLUSION

In this article, we aim to improve the understanding of challenges companies face with license compliance. To do so, we examined the mismatches between declared and in-code licenses of open source software repositories from the perspective of industry. The declared license is explicitly stated in one or more standard licensing documents, while the in-code license is found in the code by an in-depth analysis using license scanners. To this end, we sampled 1,000 GitHub repositories and extracted both the declared and the in-code licenses. To broaden this sample or to replicate this study in future work, large-scale software datasets with open source license metadata can be used, such as the newly available dataset consisting of 6.5 million unique license files published by Zacchiroli [59], who uses the ScanCode tool for license detection similar to our approach. An overview of the code used for our work is available as a Docker Container[15].

We find widespread inconsistencies between declared and actual in-code licenses. Specifically, about half of the repositories in our data set showed inconsistencies between declared and in-code licenses. This shows that open source repositories oftentimes have undeclared licenses in subdirectories. This is consistent with the findings of German et al., who found similar results in Fedora 12 source code packages [20]. Furthermore, about half of the discovered license mismatches for declared permissive licenses were with licenses of higher restrictiveness. As our work did not consider inconsistencies between in-code licenses, future work which considers this may find that license inconsistency is even more prevalent than we have identified.

Throughout our study, we also wanted to understand the origins of license mismatches. Following our analysis, we noticed that most of the inconsistencies arise from declared to in-code mismatches. Declared to declared inconsistencies do occur, but are much rarer. Thus, the problem of faulty license declarations most likely originates from the inclusion of source code from other projects, without taking note of the corresponding licenses. The possible reasons for this are beyond the scope of this paper and should be investigated further in future research. However, it is important to consider that once in circulation, these inconsistencies and conflicts can spread to other repositories that incorporate them.

As a consequence, any software company who builds systems by using open source software will often not fulfill the license requirements they have to fulfill to lawfully build on the open source software, because only the declared licenses were known. Both the seemingly innocuous attribution requirement (give credit to the original open source programmers) and the heavyweight copyleft requirement (pass on your software only under the original copyleft license) opens up the company to lawsuits by the aggrieved original open source programmers. While license scanners can be used to add assurance, this approach is imperfect and does not completely eliminate risk.

In particular our observation of copyleft-licensed software hiding behind a permissive license in about ten percent of the repositories in our sample represents a serious threat to traditional software companies whose business model relies on keeping their source code proprietary. It does not matter who (insufficiently) labeled the original open source code: A software company still has to comply with the correct license. For this reason, we see the need for a major clean-up of existing open source code and the establishment of proper governance by all open source repositories to enable users to perform their own governance. Repositories who fail to do so make compliance more difficult for anyone using the software, which may affect the project's ability to attract a community and user base. Furthermore, making improvements to the compliance process on social coding platforms, such as GitHub, could help alleviate this problem and lead to developers being more aware and conscientious of license governance during development.

[15]Docker Container with used code – https://hub.docker.com/r/wolterfau/licenseconfusion

For future work, we suggest to look deeper into the topic of open source licensing conflicts, by analyzing differences between different in-code licenses. Furthermore, more research could be conducted in the origin of mismatched in-code licenses and how to incentivise software developers to pay closer attention to open source licensing.

Acknowledgements: We would like to thank Nathalie Schnelzer and Yannik Schmidt, whose master's theses helped to identify potential areas for future research which are addressed in this paper. Additionally, Michael Dorner supported several students by providing technical assistance with infrastructure and data collection.

REFERENCES

[1] Ågerfalk, P. J., and Fitzgerald, B. Outsourcing to an unknown workforce: Exploring opensurcing as a global sourcing strategy. *MIS quarterly* (2008), 385–409.

[2] Almeida, D. A., Murphy, G. C., Wilson, G., and Hoye, M. Do software developers understand open source licenses? In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)* (2017), IEEE, pp. 1–11.

[3] Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. Analyzing software licenses in open architecture software systems. In *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (2009), IEEE, pp. 54–57.

[4] August, T., Chen, W., and Zhu, K. Competition among proprietary and open-source software firms: The role of licensing in strategic contribution. *Management Science 67*, 5 (2021), 3041–3066.

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

[6] Baltes, S., and Diehl, S. Usage and attribution of stack overflow code snippets in github projects. *Empirical Software Engineering 24*, 3 (2019), 1259–1295.

[7] Black Duck Software. 2017 Open Source Security & Risk Analysis. (2017).

[8] Coughlan, S. Standardizing open source license compliance with openchain. *Computer 53*, 11 (2020), 70–74.

[9] Crowston, K., Wei, K., Howison, J., and Wiggins, A. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR) 44*, 2 (2012). 7.

[10] Dyck, S., Haferkorn, D., Kerth, C., and Schoebel, A. Automating open source software license information generation in software projects. *Journal of Systemics, Cybernetics and Informatics 16*, 5 (2018), 44–49.

[11] European-Commission. The economic and social impact of software & services on competitiveness and innovation (smart 2015/0015). https://op.europa.eu/en/publication-detail/-/publication/480eff53-0495-11e7-8a35-01aa75ed71a11. Accessed: 2021-04-22.

[12] Fendt, O., and Jaeger, M. C. Open source for open source license compliance. In *Open Source Systems* (Cham, 2019), F. Bordeleau, A. Sillitti, P. Meirelles, and V. Lenarduzzi, Eds., Springer International Publishing, pp. 133–138.

[13] Fendt, O., and Jaeger, M. C. Open source for open source license compliance. In *IFIP International Conference on Open Source Systems* (2019), Springer, pp. 133–138.

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

[15] Fitzgerald, B. The transformation of open source software. *MIS quarterly* (2006), 587–598.

[16] Fontana, R., Kuhn, B. M., Moglen, E., Norwood, M., Ravicher, D. B., Sandler, K., Vasile, J., and Williamson, A. A legal issues primer for open source and free software projects. *Software Freedom Law Center* (2008), 1–37.

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

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

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

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

[21] German, D. M., and Hassan, A. E. License integration patterns: Addressing license mismatches in component-based development. In *Proceedings of the 31st International Conference on Software Engineering* (2009), IEEE Computer Society, pp. 188–198.

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

[23] Gobeille, R. The fossology project. In *Proceedings of the International Working Conference on Mining Software Repositories* (2008), ACM, pp. 47–50.

[24] Hall, A. J. Open-source licensing and business models: Making money by giving it away. *Santa Clara Computer & High Tech. LJ 33* (2016), 427.

[25] Harutyunyan, N. Managing your open source supply chain-why and how? *Computer 53*, 6 (2020), 77–81.

[26] Harutyunyan, N., Bauer, A., and Riehle, D. Industry requirements for floss governance tools to facilitate the use of floss components in commercial products. *Journal of Systems and Software: Under Review* (2019).

[27] Harutyunyan, N., and Riehle, D. Getting started with corporate open source governance: A case study evaluation of industry best practices. In *Proceedings of the 54th Hawaii International Conference on System Sciences* (2021), p. 6263.

[28] Hassin, K. Open source on trial. *Open Source Business Resource* (2007).

[29] Hauge, Ø., Ayala, C., and Conradi, R. Adoption of open source software in software-intensive organizations–a systematic literature review. *Information and Software Technology 52*, 11 (2010), 1133–1154.

[30] Heale, R., and Twycross, A. Validity and reliability in quantitative studies. *Evidence-based nursing 18*, 3 (2015), 66–67.

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

[32] Hemel, A., Kalleberg, K. T., Vermaas, R., and Dolstra, E. Finding software license violations through binary code clone detection-a retrospective. *ACM SIGSOFT Software Engineering Notes 46*, 3 (2021), 24–25.

[33] Jaeger, M. C., Fendt, O., Gobeille, R., Huber, M., Najjar, J., Stewart, K., Weber, S., and Wurl, A. The fossology project: 10 years of license scanning. *IFOSS L. Rev. 9* (2017), 9.

[34] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (New York, NY, USA, 2014), MSR 2014, ACM, ACM, p. 92–101.

[35] Kapitsaki, G. M., and Charalambous, G. Modeling and recommending open source licenses with findosslicense. *IEEE Transactions on Software Engineering 47*, 5 (2019), 919–935.

[36] Kapitsaki, G. M., Kramer, F., and Tselikas, N. D. Automating the license compatibility process in open source software with spdx. *Journal of systems and software 131* (2017), 386–401.

[37] Kon, F., Meirelles, P., Lago, N., Terceiro, A., Chavez, C., and Mendonça, M. Free and open source software development and research: Opportunities for software engineering. In *2011 25th Brazilian Symposium on Software Engineering* (2011), IEEE, pp. 82–91.

[38] de Laat, P. B. Governance of open source software: state of the art. *Journal of Management & Governance 11*, 2 (2007), 165–177.

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

[40] Lopes, C. V., Maj, P., Martins, P., Saini, V., Yang, D., Zitny, J., Sajnani, H., and Vitek, J. DéjàVu: a map of code duplicates on GitHub. *Proceedings of the ACM on Programming Languages 1* (October 2017).

[41] Marois, P., Marsan, J., Carillo, K., Stol, K.-J., and Fitzgerald, B. A Delphi study of obsolete assumptions in free/libre and open source software. In *Proceedings of the European Conference on Information Systems* (2022), AIS.

[42] Moraes, J., Polato, I., Wiese, I., Saraiva, F., and Pinto, G. From one to hundreds: multi-licensing in the JavaScript ecosystem. *Empirical Software Engineering 26* (05 2021).

[43] Mustonen, M. Copyleft—the economics of linux and other open source software. *Information Economics and Policy 15*, 1 (2003), 99–121.

[44] Ombredanne, P. Free and open source software license compliance: tools for software composition analysis. *Computer 53*, 10 (2020), 105–109.

[45] Popp, K. M. *Best Practices for commercial use of open source software: Business models*. BoD–Books on Demand, 2019.

[46] Riehle, D. Single-vendor open source firms. *Computer 53*, 4 (2020), 68–72.

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

[48] Rosen, L. *Open source licensing*, vol. 692. Prentice Hall, 2005.

[49] Rousseau, G., Di Cosmo, R., and Zacchiroli, S. Software provenance tracking at the scale of public source code. *Empirical Software Engineering* (2020).

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

[51] Ruffin, M., and Ebert, C. Using open source software in product development: a primer. *IEEE Software 21* (2004), 82–86.

[52] Squire, M. The lives and deaths of open source code forges. In *Proceedings of the 13th International Symposium on Open Collaboration* (New York, NY, USA, 2017), ACM.

[53] Stewart, K. J., Ammeter, A. P., and Maruping, L. M. Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. *Information Systems Research 17*, 2 (2006), 126–144.

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

[55] Tuunanen, T., Koskinen, J., and Kärkkäinen, T. Automated software license analysis. *Automated Software Engineering 16*, 3 (2009), 455–490.

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

[57] Wolter, T. A comparison study of open source license crawler. Bachelor Thesis, 2019.

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

[59] Zacchiroli, S. A large-scale dataset of (open source) license text variants. In *Proceedings of the 19th International Conference on Mining Software Repositories* (2022), IEEE.