To Distribute or Not to Distribute? Why Licensing Bugs Matter

Christopher Vendome
College of William and Mary
Williamsburg, VA

Daniel M. German
University of Victoria
BC, Canada

Massimiliano Di Penta
University of Sannio
Benevento, Italy

Gabriele Bavota
Università della Svizzera italiana (USI)
Lugano, Switzerland

Mario Linares-Vásquez
Universidad de los Andes
Bogotá, Colombia

Denys Poshyvanyk
College of William and Mary
Williamsburg, VA

ABSTRACT

Software licenses dictate how source code or binaries can be modified, reused, and redistributed. In the case of open source projects, software licenses generally fit into two main categories, permissive and restrictive, depending on the degree to which they allow redistribution or modification under licenses different from the original one(s). Developers and organizations can also modify existing licenses, creating custom licenses with specific permissive/restrictive terms. Having such a variety of software licenses can create confusion among software developers, and can easily result in the introduction of *licensing bugs*, not necessarily limited to well-known license incompatibilities. In this work, we report a study aimed at characterizing licensing bugs by (i) building a catalog categorizing the types of licensing bugs developers and other stakeholders face, and (ii) understanding the implications licensing bugs have on the software projects they affect. The presented study is the result of the manual analysis of 1,200 discussions related to licensing bugs carried out in issue trackers and in five legal mailing lists of open source communities. Our findings uncover new types of licensing bugs not addressed in prior literature, and a detailed assessment of their implications.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **Social and professional topics →** *Licensing*;

KEYWORDS

Software Licenses, Empirical Studies, Open Source Practices

ACM Reference Format:
Christopher Vendome, Daniel M. German, Massimiliano Di Penta, Gabriele Bavota, Mario Linares-Vásquez, and Denys Poshyvanyk. 2018. To Distribute or Not to Distribute? Why Licensing Bugs Matter. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180221

1 INTRODUCTION

Software licenses define the terms under which a software system (either its code or binary) can *legally* be distributed, modified, and reused. As of today, there are over 100 known open source software licenses [61, 62]. In a broad sense, licenses can be divided into two categories: *restrictive* licenses (such as the General Public License—GPL—family of licenses), requiring the redistribution of derivative work under the same licensing terms, and *permissive* licenses, which impose few restrictions (such as the MIT license). Open source communities also have their own restrictions and guidelines concerning the adoption of software licenses. For example, the Apache Software Foundation requires contributions to be licensed under the Apache-2.0 License. Similarly, Debian utilizes the "Debian Social Contract" [60] or "Debian Free Software Guidelines" (DFSG) to evaluate whether a license is free or non-free; this evaluation directly determines whether a piece of software can be bundled into Debian or must be distributed separately.

Prior empirical studies [39, 43, 45, 47, 56–58, 65, 67, 68] showed that (i) developers face difficulties in choosing the appropriate license for their work as well as in understanding the legal implications behind licenses, and (ii) incompatibilities between licenses can represent a serious threat from a legal perspective for both open source and commercial software. These two pieces of evidence combined suggest that licensing issues are likely to occur in software projects and that their effect could be non-negligible. However, as of today, there is limited evidence in the literature about the licensing issues that developers generally face and of their legal/technical implications [43]. In this work, we refer to such issues as *licensing bugs*, mostly related to legal incompatibilities of licensed software and to the breach of guidelines that can prevent software from being distributed or modified (*e.g.,* by preventing a patch from being accepted). While licensing bugs may not be as pervasive as other types of bugs, they have the ability to block development and prevent software from being released, and in the worst case, may result in monetary implications.

We present a large-scale qualitative study aimed at characterizing *licensing bugs*, with the goal of understanding the types of licensing bugs developers face, their legal and technical implications, and how such bugs are fixed. In this study, we aim at answering the following research question: *What are the types of licensing bugs faced by developers?* Specifically, we investigate the types of licensing bugs that developers discuss (and try to solve) in various sources of communications. We aim to define a catalog of licensing bugs, which includes the impacted stakeholders, the implications and ability to address the licensing bugs.

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180221

To this aim, we mined 59,426 discussions (from issue trackers and mailing lists) carried out by software developers and likely related to licensing bugs. Then, we manually analyzed — via an open-coding procedure — a statistically significant sample of 1,200 discussions, randomly selected from the initial set of 59,426 discussions. During the manual analysis, we transcribed the type of licensing bugs, their implications, and possible fixes.

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

Our proposed catalog can serve as a reference for developers and lawyers dealing with potential licensing issues. Additionally, it can help open source community refine their guidelines so that contributors have a more clear understanding of their implications. Lastly, tool designers and researchers can utilize our catalog to assist developers with open source license compliance.

2 RELATED WORK

A substantial part of the research work carried out in the software licensing field focused on the definition of techniques and tools supporting the automatic identification of licenses in software products. For instance, the FOSSology project [49] exploited machine learning techniques to automatically classify licenses, thus, supporting the task of license identification for a given software. Tuunanen *et al.* [63] presented ASLA, a reverse engineering tool supporting the identification of software licenses with an accuracy of 89%. Afterwards, German *et al.* [48] proposed *Ninka*, an approach using sentence matching and currently representing the state-of-the-art, with a precision of 95%. Di Penta *et al.* [42] proposed an approach relying upon code search in order to identify the licensing of jars. Finally, German *et al.* [44] analyzed 523,930 archives in order to understand the impact the accuracy of identifying FOSS licenses when used in conjunction with proprietary licensing.

Previous work has also analyzed the usage, evolution and inconsistencies of open source licensing in software projects. Di Penta *et al.* [43] investigated the migration of licenses over the course of a project's lifetime. Their study suggests that licenses changed version and type during software evolution, but there were no generic patterns generalizable to the six analyzed FOSS projects. German *et al.* [47] 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.* [45] presented an empirical study focused on the binary packages of the Fedora-12 Linux distribution aimed at (i) understanding if licenses declared in the packages were consistent with those present in the source code files, and (ii) detecting licensing issues derived by dependencies between packages. As a result of their investigation, German *et al.* [45] were able to find some licensing issues confirmed by Fedora. Manabe *et al.* [51] analyzed the changes in licenses of FreeBSD, OpenBSD, Eclipse, and ArgoUML, finding that each project had different evolution patterns. Vendome *et al.* [67] investigated license usage and changes in licensing to understand the rationale behind potential usage and changes. Vendome *et al.* [68] also investigated when developers pick a particular license or changes the license(s) and conducted a survey to understand the underlying perspective and rationale of developers with respect to choosing or changing their system's licensing. Almeida *et al.* conducted a survey to investigate the extent to which developers understand licenses [39]. License inconsistencies have also been analyzed in a study by German *et al.*, which considered code clones between Linux and two BSD distributions [46] as well as a study by Wu *et al*, which considered code clones in Debian 7.5 with inconsistent licensing, potentially representing license violations [69].

Most of the aforementioned work has focused on C/C++ and Java languages. However, few studies have focused on other languages, for instance, Mlouki *et al.* [54] investigated license violations and their evolution in 857 open source Android apps; 229 releases for 17 of the analyzed apps were found to have license issues, in particular, violations to the terms of open source licenses. Vendome *et al.* [66] found 14 different license exceptions (i.e., additional grants/restrictions specified by the copyright holders beyond the canonical license) in 298 files from open source projects developed in six programming languages; although license exceptions are not prevalent in open source projects, they may introduce license bugs since they modify the canonical versions of open source licenses.

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

3 EMPIRICAL STUDY DESIGN

We aim at addressing the following question: *What are the types of licensing bugs faced by developers?* We investigate licensing bugs in FOSS projects, by characterizing them in terms of types, difficulties they pose, their implications on legal and technical aspects, and the ability to (or the extent to which) licensing bugs can be addressed (or resolved). To this, we mined developers' discussions from (i) issue trackers hosted on GitHub [28] and Bugzilla [10], and (ii) mailing lists specialized in legal aspects of software products. All the data used in this study are available in the attached appendix [38].

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

3.1 Identification of Candidate Licensing Bugs from Developers' Discussions

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

We focused on mailing lists dealing with the discussion of legal aspects of software products, and in particular: Apache's *legal-discuss* [2], Debian's *debian-legal* [12], Fedora's *fedora-legal-list* [26], Gnome's *legal-list* [27], and OpenStack's *legal-discuss* [36]. All the considered mailing lists have publicly available archives storing all messages exchanged through them. Once the archives were downloaded, we automatically identified discussions linking each message to the corresponding email thread by exploiting the "Message-id" field in the "In-reply-to" field. This allowed us to analyze each message in the context of the discussion, thus favoring an easier interpretation of the legal issue being discussed. In total, we had 45,019 candidate messages with licensing bugs.

Concerning the issue trackers, we mined the 136 Bugzilla issue trackers listed in the Bugzilla's installation list [6]. To identify issues relevant to licensing, we queried the Bugzilla's search functionality using the keyword "*license*". This led to the selection of 2,125 candidate licensing bugs. Additionally, we mined the issue trackers of 86,032 projects hosted on GitHub. We selected projects that (i) are not a fork (to avoid duplications), and (ii) have at least one star, watcher, or fork (to perform a first cleaning of "toy" projects) [50]; we then locally cloned these repositories and filtered out all projects with less than fifty commits, again with the goal of removing trivial projects. Out of the selected 258,057 projects, we considered 86,032 projects that had an issue tracker. In particular, we crawled the candidate licensing-related issues (including the issue title, description, metadata, and related discussion) from each issue tracker by using a keyword search mechanism exploiting specific licensing keywords (*e.g., copyright*) or license names (*e.g., GPL*) [67]. This keywords-based search resulted in 12,282 candidate licensing-related issues.

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

3.2 Definition of the Catalog of Licensing Bugs

To devise our catalog of bugs, we manually inspected the 1,200 selected mailing list and issue tracker discussions (from now on, generally referred to as "discussions") via an open-coding procedure [53]. The goal was to categorize the type of licensing bugs. We divided the 1,200 documents among the authors such that each document was coded by at least two-authors for two-author agreement, obtaining a list of 21 initial categories (each discussion belonging to a single category). While we had attempted to filter data to remove false positives, there were still 78 documents identified as false positives (6.5%) and 21 tagged as unclear (1.83%). These two categories are omitted from the presented catalog in the results.

Each discussion was randomly assigned two of the authors to independently categorize it. After each round, the two authors discussed and resolved conflicts, and generated categories from the codings. Before solving conflicts, the coding achieved an agreement ratio of 65%. To determine whether such agreement level was due to chance, we computed the Cohen's kappa inter-rater agreement coefficient [41], which resulted to 0.53 (moderate agreement).

After, we defined a higher level of abstraction over the set of 21 initial distinct categories using a card-sorting approach [59]. To this aim, the 21 categories were presented in a spreadsheet, and two of the authors independently reordered the categories and clustered them. After that, they discussed the clusters they had created, and converged to a common solution. In the end, this resulted in grouping license bugs into seven distinct categories, listed in the 21 sub-categories previously identified (Table 1).

4 CATALOG OF LICENSING BUGS

We present and discuss the catalog of licensing bugs obtained by following the procedure described in Section 3. Specifically, we present a description of each licensing bug, the stakeholders *most directly impacted* by the particular licensing bug (i.e., the individuals most directly responsible for them and their remediation), the implications (both legal and technical) of the bugs, and the extent to which they can be addressed/fixed. Also, we provide examples of discussions related to that particular category of licensing bug. While previous studies on licensing [68] mostly focused on developers, we consider different stakeholders impacted by the bugs:

(1) **Integrators:** developers that are reusing open source software within their own systems;
(2) **Package Maintainers:** developers responsible for maintaining packages and integrating patches or bug-fixes into existing distributions of software;
(3) **Distributors:** any individual or entity distributing software, which can be a single developer, an open source foundation/-community (*e.g.,* Apache), or a company (*e.g.,* IBM);
(4) **Developers:** this category relates to developers in general, without making specific distinctions;
(5) **Lawyers:** interpreting licensing in terms of the existing laws or responsible for drafting new licenses;
(6) **Lawmakers:** people responsible for writing and passing laws;
(7) **Community:** either people involved in a specific open source community, or the open source community as a whole;
(8) **Trademark Holders:** companies or individuals that have trademarked their name or logo (*e.g.,* as a means of brand protection).

Table 1: Catalog of Licensing Issues with (# of Discussions).

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

The taxonomy is composed of 21 distinct sub-categories organized in 7 distinct high-level categories (Table 1). Due to space limitations, we only discuss a subset of the sub-categories (14). The complete taxonomy description and frequencies of each category can be found in the attached appendix [38]. In the reported examples, we have adjusted formatting (*e.g.,* removing mid-sentence newlines) in some cases, but the wording remains unchanged. It is important to remark that the results discuss the interpretation of developers and/or legal practitioners. Therefore, it is possible that the legality of these interpretations or discussions may change (*e.g.,* new interpretations can cause new legal precedents in the U.S.A.), or the enforceability may change in different jurisdictions. Our results have the purpose of discussing these interpretations and implications from the perspective of the developers and legal practitioners within particular open source communities.

4.1 Laws and Their Interpretations

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

4.1.1 What is Copyrightable?

Description: Developers experience issues when trying to understand the implications or scope of copyright coverage. We observed discussions regarding copyright and emulation, legal consequences of game console emulators, executed programs, or donated code. We also observed discussions related to textual updates to copyrights, *e.g.,* copyright year upgrade. We found that the scope of the coverage is not necessarily clear for developers. While software is copyrightable (the basis of free and open source software licensing), higher-level design and ideas *may* fall out of the scope of copyright law. Potential disagreements on the coverage of copyright protection can potentially lead to the infringement of another entity's copyright. In an example below, we demonstrate the difficulty to understand the coverage of copyright for game emulation.
Stakeholders: Developers and lawyers.
Implications: It has been demonstrated that copying as few as 27 lines of code can constitute copyright infringement [52]. Violating these laws can result in legal action against developers and prevent the distribution or reuse of the system violating the copyright (see the subsection on *License Violation*). However, the scope of copyright law (similar to patent and trademarks) can only be addressed by lawyers, and to truly define the scope it would require lawmakers to more explicitly or rigidly define the extent of copyright; alternatively, in precedent-based legal systems, prior court rulings may also serve to define the scope and implications of copyright.
Examples: In terms of emulation, one individual commented in the thread that emulation of the games themselves can be illegal,

> "That second case is pretty much where we stand with a *lot* of game console emulators out there – the only way to get data to use with them is to break the law. Wonderful." [23]

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

> "Is it illegal if I own a game cartridge, and dump it? That part probably isn't; US copyright law, at least, give me permission to make a backup copy." [23]

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

4.1.2 What is a Derivative Work?

Description: A derivative work is partially owned by the copyright author on which it is originally based. In order to be able to distribute a derivative work, its creator needs a license from the work on which it is based. In fact, one of the most important features of open source licenses is that they should allow the creation and redistribution of derivative works. However, some licenses place important restrictions on such redistribution (*e.g.,* the GPL family of licenses requires that any derivative work must also be distributed under the GPL). Therefore, when a software system reuses another product, the question on whether the former is a derivative work of the latter is critical. This issue might affect not only software, but also non-software artifacts, like images.
Stakeholders: Distributors, package maintainers, integrators.
Implications: If a product *A* is a derivative work of product *B*, the license of product *B* can impose restrictions on how product *A* can be used and licensed to others. Such restrictions could mean that product *A* might not be used or licensed as expected. Or, if product *A* is redistributed, it could result in legal liability due to the risk of copyright infringement. On the other hand, if product *A* is not a derivative work of product *B*, it might be possible to redistribute product *B* along product *A*, as long as the license of product *B* is properly satisfied. Thus, evaluation of whether a work is considered derivative work can either result in license violations or incompatibilities if the derivative work is considered non-free or subject to the terms of an incompatible restrictive license. To cope

with such bugs, when possible developers can use/ask for license exceptions [66].

Examples: Some systems provide clarifications on whether something is considered a derivative work of another. For example, there has been a long discussion on what is a derivative work of the Linux kernel. Linus Torvalds clarifies this in the COPYING file,

> "NOTE! This copyright does not cover user programs that use kernel services by normal system calls - this is merely considered normal use of the kernel, and does not fall under the heading of "derived work". Also note that the GPL below is copyrighted by the Free Software Foundation, but the instance of code that it refers to (the Linux kernel) is copyrighted by me and others who actually wrote it." [34]

However, there is still plenty of disagreement on this issue [64].

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

> "However – by this definition, the linux kernel is very definitely a derivative work, and the firmware is content which has been incorporated into the kernel." [18]

The developer that presented the legal text follows up saying:

> "The kernel (I assume as a whole) is a derivative work of what? I would argue that the kernel is a compilation of what executes on the host CPU with other parts (boot logos, fonts, and firmware data). The executable part may be a derivative in part of Adam Richter's code, but that does not necessarily make the kernel as a whole a derivative of his work." [19]

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

4.1.3 *What is the Jurisdiction?*

Description: We observed licensing bugs related to the differences in laws across countries. Copyright, trademark and patent laws are national in scope. While the World Intellectual Property Organization (WIPO) attempts to unify intellectual property law around the world, specific issues of the law might be different from one country to another. Also, software is affected by other laws, such as restrictions on trade. For example, moral rights are enshrined in the copyright laws of Europe and Canada, but they are not present in the copyright laws of the United States.

Stakeholders: Distributors, developers, and lawmakers.

Implications: These differences might not be large, but they might have an important impact on the ability to create and distribute software.They might also have an impact on any potential litigation (and the choice of jurisdiction). In fact, we observed that clauses related to choice of jurisdiction were a controversial topic within Debian in terms of their impact on software's freeness. However, the distribution may be impacted by external factors like trade restrictions to a particular country or distribution of what a country considers sensitive material. While organizations or communities may want to facilitate global reuse, the organizations and individuals must comply with these trade laws. Also, government funded work (*e.g.,* supported by a grant) may require release of the software for public domain domestically (*i.e.,* within the funding country), but the government may impose restrictions internationally, which can only be addressed by policy changes by the funding entity. Thus, these cases are predominantly external to developers, but impose restrictions that must be considered.

Examples: In *debian-legal*, there is question regarding a texture created by NASA and the possibility to reuse the texture. While one person indicates that the copyright would depend on whether it was done by NASA itself or a contractor (*e.g.,* a contractor of the Jet Propulsion Laboratory), another person comments:

> "While a work may be in the public domain in the U.S., it may be under copyright elsewhere. So, *e.g.,* while works by the U.S. government may be public domain in the U.S., they may remain under copyright in other countries. However, the U.S. government may license their works and thus give permissions with respect to these foreign copyrights." [17]

Thus, the copyright issues become further complicated by the fact that government created works might only be released domestically into the public domain [1].

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

> "It is my understanding that it is illegal under US law to knowingly export software (free software or otherwise) from the US to Cuba, or to Cuban nationals." [14]

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

4.2 Policies of the Ecosystem

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

4.2.1 *Community Guidelines.*

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

Stakeholders: Community and lawyers.

Implications: Issues in this category have wide reaching repercussions and impact on a community. Unlike other licensing bugs, these reflect the collaborative effort within a community to enforce a particular licensing policy. The community needs to protect itself from legal liability of distributing potentially incompatible code, while also forging collaborations between the open source community and corporations. Thus, lawyers are also important in designing the guidelines to legally protect the community from potentially violating licensing. The agreed upon guidelines will have

a long term impact in the community and will serve as a contract that specifies what licensing actions are appropriate and which are not. As a consequence, they have the potential to restrict the pool of software that can be reused/integrated into a new product.
Examples: We observe an issue with Eclipse's community restrictions on software licensing. To circumvent the restriction, end-users assume the burden to properly configure their environment to utilize that software. The developer states:

> "A number of projects would like to integrate with external libraries under licenses not compatible with the EPL (LGPL, ...). What can be done in order to help end-users come up with a working environment easily? Clearly, * The LGPL library must be installed from a source hosted outside Eclipse * A CQ must be filed indicating the dependency as "works-with" or "requires" But how can end-users or other clients be instructed to get the complete environment going?" [7]

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

4.2.2 *Freeness: can I reuse it?*

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

This category affects the integration of many types of artifacts, such as source code, images (icons, logos), databases, text files, etc. Ultimately, the creation and enactment of similar guidelines by anybody developing software (whether open source or not) provides an opportunity for an up-front discussion on what legal risks an organization is willing to take, and, as a consequence, the pool of artifacts that can be reused.
Stakeholders: Integrators, package maintainers, & distributors.
Implications: In the case of Debian and Fedora, software with non-free licenses cannot be included in the main distribution and must be distributed separately. We observe one type of discussion relating to the freeness of open source licenses, in particular the GPL, QPL, AGPL, and Artistic License. Additionally, it is important that the entire system meets the community guidelines, including the needed dependencies. However, source code and binaries are not the only artifacts evaluated. Since IP clearance/evaluation extends to all bundled artifacts (not only source code and binaries), a *non-free* image or font could prevent the distribution of the software. Last, but not least, the community discusses specific clauses concerning how to deal with "Freeness" requirements.
Examples: We observe in RedHat's Bugzilla tracker that a license of *mpage* prevents source code modification, which makes the license non-free. The reporter of the bug states in the first two comments on the issue tracker:

> "mpage included non-free code, Please see Debuan [sic] bug number "805370" details. I think that this package be affected by debian bug number "805370"...Blocking FE-Legal, This is license problem." [8]

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

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

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

> "Whereas the alternative may be that licensors are unable to afford the enforcement of their license. Would you prefer to discriminate against them?" [13]

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

4.3 Potential License Violations

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

4.3.1 *Compatibility between Licenses.*

Description: This category is related to issues relating to reusing software with different licenses. These issues arise when a developer utilized dependencies or reused source code that is not compatible with either the declared license, or with the license of other reused components. We observe incompatibilities between the license of dependencies to the systems reusing (or seeking to reuse) them. Additionally, a dependent artifact may change its licensing, possibly introducing a licensing incompatibility. These bugs also discuss the compatibility of licenses to understand under what condition software under two licenses can be used together (if all). These compatibility issues may also be related to a community's standards, *e.g.,* Apache Software Foundation requiring the Apache Software License for submitted contributions.
Stakeholders: Integrators, package maintainers, and distributors.
Implications: These incompatibility issues impact the ability to distribute the software, since at least one license of reused software is being violated from the incompatibility. These issues will either stall a patch or prevent a submitted system from being distributed within a community. These bugs can require substantial effort to fix, depending on the available replacement options for the violating code or the ability to migrate the software's license in order to be compatible with the reused code. To address these issues, the original developers need to remove the incompatible code/binary and find an alternative implementation that is licensed under a compatible license. In the case of a community's standard, it can also be problematic, since migrating the system's license can introduce other violations between dependencies or violate the community guidelines. For example, a project belonging to the Apache Software Foundation cannot migrate towards the GPL license. Additionally, a license change can prevent newer releases of a certain library from being integrated and forcing developers to rollback to a compliant

release of that library. Thus, these issues are not always trivial to fix and can cause a longer delay for a contribution to be accepted.
Examples: In an issue on GitHub, a developer comments:

> "As mentioned in twbs/bootstrap#2054 as well as here, Bootstrap can not be used within any GPLv2 project as the Apache License 2.0 is incompatible. This just needs to be re-licensed under the GPLv3 to resolve this issue."[5]

The project's owner acknowledges that the license will be updated during the current rewriting of part of the code base. Interestingly, another respondent follows up saying that the compatibility is based on the Free Software Foundation's interpretation and the Apache Software Foundation differs; however, the assertion is either misinterpreted or outdated as the Apache Software Foundation does not assert compatibility with GPL-2.0 [31].

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

> "Be advised that at least 5 Apache projects appear to depend on a third party library known as "greenmail" which advertises itself in may [sic] places as being licensed under the ASL 2.0, however most of the greemail [sic] source files contain headers claiming LGPL" [3]

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

4.3.2 Other Types of License Violations.

Description: This category addresses issues that are considered license violations but do not fall into the prior category. These bugs are serious in nature and usually require removing or rewriting code. Otherwise, it might not be possible to continue distributing the product. These license violations can also be related to several other categories in our catalog, *e.g.,* incorrect licensing of derivative work or incorrect interpretations of licensing clauses. License violations can apply beyond software licensing with potential violations of the End User License Agreement (EULA).
Stakeholders: Distributors, integrators, and developers.
Implications: License violations prevent software, whether it is a patch or an entire system, from being distributed. By violating the license, it amounts to illegally distributing copyrighted material. These violations can only be addressed by the original developer, since the infringing code must be removed, or the software's license needs to be changed (if that addresses the violation). In fact, a Munich district court's injunction prevented the distribution of Fortinet Ltd's software until complying with the GPL [33]. Thus, it can directly impact a person or company's ability to distribute their project, and it could result in other civil penalties. Therefore, these licensing bugs are among the most severe as they directly impact software distribution and can involve a large technical effort to remedy the violation.
Examples: We observed a severe violation where it appears that a developer stole code and removed the original author from the copyright statement of the file(s). The author posts to the GitHub issue tracker stating:

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

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

We observed an email thread where ICQ's EULA prohibits the connection of third-party software and a developer suggests removing the third-party applications from inclusion in Fedora, stating:

> "...Thus, keeping in mind, that there is no way to use ICQ-related software w/o explicitly violating their license agreement, and there are many other open alternatives (and even proprietary systems, which permits 3rd party applications), I think that software, designed to work with explicit requirement to be connected to ICQ network, should be considered as unacceptable for inclusion into Fedora"[25]

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

4.4 Non-Source Code Licensing

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

4.4.1 Documentation Licensing.

Description: Documentation, like source code, is also protected by copyright. Hence, it is necessary to license it. Open source licenses must make sure that the documentation can also be modified and redistributed. While software licenses were designed with source code in mind, we have observed that they are frequently used to license documentation. Although developers can utilize the GNU Free Documentation License [29] or a Creative Commons license [11], not all of the Creative Commons licenses may be considered free (*e.g.,* Debian indicated that the Creative Commons Attribution License version 1.0 is not compatible with the Debian Free Software Guidelines), which can cause issues within a community from distributing the documentation.
Stakeholders: Distributors.
Implications: These licensing bugs could prevent a package from being accepted for redistribution by a community, since the redistribution would either constitute distributing a copyrighted work or distributing an incompatibly licensed work (incompatible in the sense of community standards). Migrating the documentation's license may be the easiest way to address these types of bugs. However, we also observe a potential license violation for documentation, when the *source* of the documentation is not available. While the documentation itself is available, it may not be modifiable according to the terms of the software license.
Examples: We observe an issue where the documentation (licensed under the GPL) is provided as a set of HTML files. However, these files were automatically generated. The individual states:

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

> "...no source code is provided for the .html documentation files. The GPL is explicit on the definition of source code: 'The source code for a work means the preferred form of the work for making modifications to it.' The term 'source code' has a precise meaning within the GPL, which differs slightly from the everyday use of the term. The html files are not source code as the term is used in the GPL license, because the documentation is maintained in a different form, and converted to html. Html files, while human-readable, are extremely inconvenient to modify. They are certainly not the 'preferred form' for making modifications." [24]

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

> "I don't see how Trolltech are violating anything though, since they own the copyright. They just aren't being particularly useful, so we can't redistribute the offending html documents"[22]

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

4.4.2 *Font and Media Licensing.*

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

Stakeholders: Distributors.

Implications: We observed that the licensing of these artifacts could also impact the ability to distribute a system. Indeed, these non-source/non-binary artifacts are still subject to community guidelines compliance, which developers may not realize initially. In addition, the implications of software licensing is not as clear in the context of non-source or non-binary entities. For example, interpreting the concept of *source code* (*e.g.,* the GPL-3.0 defines it as "the preferred form of the work for making modifications to it" [30]) is less clear in the context of audio (*i.e.,* does the audio file itself qualify or does it require a different representation as well). To address these issues, non-free fonts or images might have to be removed, but this also requires developers to find replacements.

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

> "Unless the creators of the podcast directly edit the MP3–which is rather unlikely–the MP3 is not the preferred form for modification and putting the MP3 under GPL without releasing the raw audio files grants no rights at all. GPLing video has a similar problem." [15]

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

> "It's a bug. If the original author puts a video under GPL and doesn't release the "source", you can't demand it. He's not bound by the GPL since he can't violate the copyright on his own work, so he has no obligation to give you anything."[16]

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

4.5 Licensing Content

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

4.5.1 *License Inconsistencies.*

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

Stakeholders: Distributors and package maintainers.

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

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

> "To fix: Licensing is both MIT/BSD and GPLv2 and GPLv2+, latter two are missing in spec file and in %license."[9]

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

4.6 Other Intellectual Property Issues

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

4.6.1 *Rights to Use a Contribution.*

Description: Many software systems, in order to clarify the provenance of their code have instituted the requirement that any contribution should be accompanied with a CTA or a CLA [55]. Contributor License Agreements (CLAs) and Copyright Transfer Agreements (CTAs) are important for intellectual property (IP) reviews. A CLA stipulates the terms of contribution (*e.g.,* the legal right to submit the code). We observed three different types of bugs in this category: (i) CLA creation, where the project has not established a CLA for contributors, (ii) CLA changes, where the CLA itself needs to be amended, (iii) missing CLA, where a contributor has not submitted a CLA. Alternatively to a CLA, CTAs transfer the ownership of the copyright from the original author (or current copyright holder) to another person(s) or legal entity.

Stakeholders: Package maintainers and integrators.

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

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

> "If you're interested in clean IP, you'll want a CLA before accepting any contributions – otherwise, you may encounter issues of people contributing code that is not theirs to contribute. For example, see the Dojo CLA. " [4]

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

4.6.2 Patent-Related Issues.

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

Stakeholders: Lawyers and distributors.

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

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

> "It's not possible to know about what patents cover a work of software because: (a) the existence of a patent is secret until it's approved (b) software is just an abstract collection of energy – it's what it symbolizes that makes it tread on a patent, or not (c) patents are written in a fashion which does not make their applicability to a work of software immediately evident." [20]

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

4.6.3 Trademark.

Description: According to the United States Copyright Office, "A trademark protects words, phrases, symbols, or designs identifying the source of the goods or services of one party and distinguishing them from those of others." [32]. Primarily, we observe the discussion related to trademarked names and logos, such as the Debian logo and package/bundle names.

Stakeholders: Distributors, Communities & Trademark Holders.

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

Examples: For example, Mozilla allows for their Firefox and Thunderbird trademarks to be used to identify *unmodified* official releases [35]. Organizations can impose different requirements regarding trademark usage, but it can be seen as a mechanism to guarantee a level of quality similar to the original system [40].

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

> "Over the years there have been a large number of packages in the archive for software whose upstream has a trademark on their name, none of whom have granted open-ended licenses to use these trademarks. We nevertheless have always made a practice of using those names unmodified as package names and binary names, on the grounds that these are interfaces that are *not subject to trademark*. This is why, whereas RedHat ships packages of 'httpd', Debian has always had packages of 'apache' even though the Apache trademark license clearly states that modified versions of the software may not use the mark. The trademark license is only relevant if we're doing something that's in scope for trademark law in the first place!" [21]

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

4.7 Licensing Semantics

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

4.7.1 License/Clause Implications.

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

Stakeholders: Developers and Integrators.

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

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

> "In fact, you should not trust any third party (even if it is FSF) about future modifications of the license which have not been taken into account by the copyright holders." [37]

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

5 LESSONS LEARNED

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

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

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

6 THREATS TO VALIDITY

Threat to *construct validity*: we mainly base our assessment and classification of licensing-related bugs on what is stated in mailing lists and issues. Nevertheless, it is possible that what is reported in such a written communication may be incomplete and imprecise.

Threats to *internal validity*: we mitigated this threat by (i) using a pre-coding phase to agree on the coding criteria, (ii) having multiple coders and a discussion process to solve cases of inconsistent coding, and (iii) computing inter-rater agreement statistics to determine whether agreements are due to chance.

Threats to *external validity*: while on one hand we have analyzed different data sources (mailing lists, Bugzilla and GitHub issue trackers) coming from different ecosystems, it is possible that the catalog we have created depends on the particular issues discussed for the studied projects, and that in other contexts developers could discuss issues we did not encounter.

7 CONCLUSIONS

We analyzed 1,200 documents from issue trackers and legal mailing lists to devise a catalog of licensing bugs containing issues developers and other stakeholders face with respect to licensing. We present descriptions of the bugs, the individuals they impact, their implications, and the ability to address them. The set of licensing bugs is diverse, which suggests that more effort should be devoted to support practitioners when dealing with licensing-related issues. We observed that licenses are also utilized with non-source code files such as fonts. While we found examples of license violations and incompatibilities, we also observed that communities may impose stricter guidelines to evaluate the freeness of a license, as done by Debian with the DFSG. Thus, compatibility relates to both the licenses themselves and the community constraints.

This project is supported in part via NSF CAREER CCF-1253837 and CCF-1525902.

REFERENCES

[1] 2017. 17 USC 105: Subject matter of copyright: United States Government works http://uscode.house.gov/view.xhtml?hl=false&edition=prelim&req=granuleid%3AUSC-prelim-title17-section105. (2017).

[2] 2017. Apache legal-discuss mail list. legal-discuss@apache.org. (2017).

[3] 2017. Apache legal message: "greenmail" library purported to be ASL may actually be LGPL; is a dependency in several ASF projects https://issues.apache.org/jira/browse/LEGAL-206. (2017).

[4] 2017. Appframework issue 21. https://github.com/01org/appframework/issues/21. (2017).

[5] 2017. bootstrap-admin issue: Incompatible License https://github.com/aristath/bootstrap-admin/issues/12. (2017).

[6] 2017. Bugzilla installation list. https://www.bugzilla.org/installation-list/. (2017).

[7] 2017. Bugzilla issue: [ip] Best Practices for interfacing with libs that are not EPL compatible https://bugs.eclipse.org/bugs/show_bug.cgi?id=246945. (2017).

[8] 2017. Bugzilla issue: mpage included non-free code https://bugzilla.redhat.com/show_bug.cgi?id=1295170. (2017).

[9] 2017. Bugzilla issue: Review Request: limnoria - A modified version of Supybot (an IRC bot) with enhancements and bug fixes https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=1342747. (2017).

[10] 2017. Bugzilla. https://www.bugzilla.org/. (2017).

[11] 2017. Creative Commons Licenses. https://creativecommons.org/licenses/. (2017).

[12] 2017. Debian-legal mail list. https://wiki.debian.org/DebianLegal. (2017).

[13] 2017. Debian legal message: Re: CDDL, OpenSolaris, Choice-of-venue and the star package ... https://lists.debian.org/debian-devel/2005/09/msg00440.html. (2017).

[14] 2017. Debian legal message: Re: Debian and Cuba https://lists.debian.org/debian-legal/2005/03/msg00493.html. (2017).

[15] 2017. Debian legal message: Re: Debian-approved creative/content license? https://lists.debian.org/debian-legal/2007/03/msg00065.html. (2017).

[16] 2017. Debian legal message: Re: Debian-approved creative/content license? https://lists.debian.org/debian-legal/2007/03/msg00069.html. (2017).

[17] 2017. Debian legal message: Re: Help about texture inclueded in stellarium https://lists.debian.org/debian-legal/2004/07/msg00832.html. (2017).

[18] 2017. Debian legal message: Re: How long is it acceptable to leave *undistributable* files in the kernel package? https://lists.debian.org/debian-legal/2004/06/msg00381.html. (2017).

[19] 2017. Debian legal message: Re: How long is it acceptable to leave *undistributable* files in the kernel package? https://lists.debian.org/debian-legal/2004/06/msg00383.html. (2017).

[20] 2017. Debian legal message: Re: IBM Jikes license. https://lists.debian.org/debian-legal/1998/12/msg00107.html. (2017).

[21] 2017. Debian legal message: Re: Packaging the MeeGo stack on Debian - Use the name ?. https://lists.debian.org/debian-legal/2011/01/msg00011.html. (2017).

[22] 2017. Debian legal message: Re: Trolltech GPL violation? https://lists.debian.org/debian-legal/2006/01/msg00011.html. (2017).

[23] 2017. Debian legal message: Re: Visualboy Advance question. https://lists.debian.org/debian-legal/2004/06/msg00619.html. (2017).

[24] 2017. Debian legal message: Trolltech GPL violation? https://lists.debian.org/debian-legal/2006/01/msg00000.html. (2017).

[25] 2017. Fedora legal message: Re: [Fedora-legal-list] Status of ICQ-related apps in Fedora. https://mid.mail-archive.com/legal@lists.fedoraproject.org/msg00185.html. (2017).

[26] 2017. Fedore-legal list. https://www.redhat.com/archives/fedora-legal-list/. (2017).

[27] 2017. Genome-legal list. https://mail.gnome.org/archives/legal-list/. (2017).

[28] 2017. Github. https://github.com/. (2017).

[29] 2017. GNU Free Documentation License. https://www.gnu.org/licenses/fdl-1.3.en.html. (2017).

[30] 2017. GNU General Public License. http://www.gnu.org/licenses/gpl.html. (2017).

[31] 2017. GPL compatibility. http://apache.org/licenses/GPL-compatibility.html. (2017).

[32] 2017. https://www.copyright.gov/help/faq/faq-general.html. (2017).

[33] 2017. An injunction against Fortinet for GPL violations. https://lwn.net/Articles/132143/. (2017).

[34] 2017. Linux Copying File: https://github.com/torvalds/linux/blob/master/COPYING. (2017).

[35] 2017. Mozilla Trademark Policy for Distribution Partners. https://www.mozilla.org/en-US/foundation/trademarks/distribution-policy/. (2017).

[36] 2017. OpenStack legal-discuss. http://lists.openstack.org/cgi-bin/mailman/listinfo/legal-discuss. (2017).

[37] 2017. picotcp issue: Upgrade license to GPLv2+ or GPLv3+ https://github.com/tass-belgium/picotcp/issues/408. (2017).

[38] 2017. To Distribute or Not to Distribute? Why Licensing Bugs Matter - Appendix. (2017).

[39] D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye. 2017. Do Software Developers Understand Open Source Licenses?. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 1–11. DOI:http://dx.doi.org/10.1109/ICPC.2017.7

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

[41] J Cohen. 1960. A coefficient of agreement for nominal scales. *Educ Psychol Meas.* 20 (1960), 37–46.

[42] Massimiliano Di Penta, Daniel M. Germán, and Giuliano Antoniol. 2010. 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.* 151–160.

[43] Massimiliano Di Penta, Daniel M. Germán, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2010. 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.* 145–154.

[44] Daniel M. Germán and Massimiliano Di Penta. 2012. A Method for Open Source License Compliance of Java Applications. *IEEE Software* 29, 3 (2012), 58–63.

[45] Daniel M. Germán, Massimiliano Di Penta, and Julius Davies. 2010. 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.* 84–93.

[46] Daniel M. Germán, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2009. 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.* 81–90.

[47] Daniel M. Germán and Ahmed E. Hassan. 2009. 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.* 188–198.

[48] Daniel M. Germán, Yuki Manabe, and Katsuro Inoue. 2010. 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.* 437–446.

[49] Robert Gobeille. 2008. 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.* 47–50.

[50] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014).* 92–101.

[51] Yuki Manabe, Yasuhiro Hayase, and Katsuro Inoue. 2010. Evolutional analysis of licenses in FOSS. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), Antwerp, Belgium, September 20-21, 2010.* 83–87.

[52] Nancy J. Mertzel. 2008. Copying 0.03 percent of software code base not 'de minimis'. *Journal of Intellectual Property Law & Practice* 3, 9 (2008), 547. DOI:http://dx.doi.org/10.1093/jiplp/jpn130

[53] Matthew B. Miles and A. Michael Huberman. 1984. *Qualitative Data Analysis: A Sourcebook of New Methods.* Sage, Beverly Hills, CA.

[54] O. Mlouki, F. Khomh, and G. Antoniol. 2016. On the Detection of Licenses Violations in the Android Ecosystem. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 382–392. DOI:http://dx.doi.org/10.1109/SANER.2016.73

[55] Germán Poo-Caamaño and Daniel M. German. 2015. The Right to a Contribution: An Exploratory Survey on How Organizations Address It. In *11th International Conference on Open Source Systems (OSS) (Open Source Systems: Adoption and Impact)*, Vol. AICT-451. 157–167. Part 5: Intellectual Property and Legal Issues.

[56] Param Singh and Corey Phelps. 2009. Networks, Social Influence, and the Choice Among Competing Innovations: Insights from Open Source Software Licenses. *Information Systems Research* 24, 3 (2009), 539–560.

[57] Manuel Sojer, Oliver Alexy, Sven Kleinknecht, and Joachim Henkel. 2014. Understanding the Drivers of Unethical Programming Behavior: The Inappropriate Reuse of Internet-Accessible Code. *J. of Management Information Systems* 31, 3 (2014), 287–325.

[58] Manuel Sojer and Joachim Henkel. 2010. Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems* 11, 12 (2010), 868–901.

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

[60] SPI. 2017. Debian Social Contract. https://www.debian.org/social_contract. (2017).

[61] The Free Software Foundation. 2017. Various Licenses and Comments about Them. https://www.gnu.org/philosophy/license-list.html. (2017).

[62] The Open Source Initiative. 2017. Open Source Licenses. http://opensource.org/licenses/category. (2017).

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

[64] Sam Varghese. 2017. Linux developer loses GPL suit against VMware http://www.itwire.com/open-source/

74288-linux-developer-loses-gpl-suit-against-vmware.html. (2017).

[65] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. 2016. License usage and changes: a large-scale study on gitHub. *Empirical Software Engineering* (2016), 1–41.

[66] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. 2017. Machine Learning-Based Detection of Open Source License Exceptions. In *9th IEEE/ACM International Conference on Software Engineering (ICSE'17)*.

[67] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. Germán, and Denys Poshyvanyk. 2015. 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, 31–40.

[68] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. 2015. 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*. IEEE, 31–40.

[69] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. Germán, and Katsuro Inoue. 2015. 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.