Understanding the Usage, Impact, and Adoption of Non-OSI Approved Licenses

Rômulo Meloca[1], Gustavo Pinto[2], Leonardo Baiser[1], Marco Mattos[1],
Igor Wiese[1], Ivanilton Polato[1], Daniel M German[3]
[1]Technological University of Paraná (UTFPR), [2]University of Pará (UFPA), [3]University of Victoria

ABSTRACT

The software license is one of the most important non-executable pieces of any software system. However, due to its non-technical nature, developers often misuse or misunderstand software licenses. Although previous studies reported problems related to licenses clashes and inconsistencies, in this paper we shed the light on an important but yet overlooked issue: the use of non-approved open-source licenses. Such licenses claim to be open-source, but have not been formally approved by the Open Source Initiative (OSI). When a developer releases a software under a non-approved license, even if the interest is to make it open-source, the original author might not be granting the rights required by those who use the software. To uncover the reasons behind the use of non-approved licenses, we conducted a mix-method study, mining data from 657K open-source projects and their 4,367K versions, and surveying 76 developers that published some of these projects. Although 1,058,554 of the project versions employ at least one non-approved license, non-approved licenses account for 21.51% of license usage. We also observed that it is not uncommon for developers to change from a non-approved to an approved license. When asked, some developers mentioned that this transition was due to a better understanding of the disadvantages of using an non-approved license. This perspective is particularly important since developers often rely on package managers to easily and quickly get their dependencies working.

CCS CONCEPTS

• **Software and its engineering → Open source model**;

KEYWORDS

Open Source Software, Software license, OSI approved

ACM Reference Format:
Rômulo Meloca[1], Gustavo Pinto[2], Leonardo Baiser[1], Marco Mattos[1], Igor Wiese[1], Ivanilton Polato[1], Daniel M German[3]. 2018. Understanding the Usage, Impact, and Adoption of Non-OSI Approved Licenses. In *Proceedings of MSR '18: 15th International Conference on Mining Software Repositories , Gothenburg, Sweden, May 28–29, 2018 (MSR '18),* 11 pages.
https://doi.org/10.1145/3196398.3196427

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196427

1 INTRODUCTION

The software licenses are one of the most important non-executable part of any software system [5]. Particularly relevant to open-source software (OSS), open-source licenses not only drive how one can use an OSS but also ensure to what extent others can reuse it [19]. Similarly to software code, software licenses change [27] and evolve [25]. Software relicensing is, indeed, commonplace in open-source software world [7]. As an example, Facebook recently relicensed four key open-source softwares from BSD + Patents to MIT license[1]. According to them, this change was motivated by an unhappy community looking for alternatives under permissive licenses. This concern, however, pertains not only to large software companies that maintain open-source softwares, since software license is a common good of any open-source software. Therefore, there is no surprise that software licensing is an active research field [1, 4, 16, 23].

Despite of its importance, developers do not fully understand problems related to license usage [1], such as the lack of licenses or license inconsistencies. The way developers develop software only exacerbates this problem, since simple actions such as copying a code snippet from the web has the potential of infringing a software license [12, 13]. This issue becomes even more relevant in the open-source era, where a constant flow of new open-source software is born at a regular basis [10]. That is, developers have a myriad of codebases to refer to, but the way they do might infringe a software license (and consequently the whole chain of software that depends on it).

Another relevant yet not fully understood problem is the use of open-source licenses that have not been approved by OSI, the Open Source Initiative (see Section 2 for details). Such software licenses were not formally approved by an open-source regulator and, therefore, has not been vetted to be open-source. Currently, OSI maintains a list with 83 approved open-source software licenses[2]. All these licenses went through a rigorous review process, and not all licenses submitted are approved (e.g., the CC0 license[3] has been submitted but was not approved). According to their website, the purpose of the OSI's license review process is to "(1) Ensure approved licenses conform to the Open Source Definition (OSD), (2) Identify appropriate License Proliferation Category, (3) Discourage vanity and duplicative Licenses"[4]. Furthermore, because OSI defined what open source is (the Open Source Definition) it claims that "only software licensed under an OSI-approved Open Source license should be labeled 'Open Source' software."[5]

[1]https://code.facebook.com/posts/300798627056246
[2]https://opensource.org/licenses/alphabetical
[3]https://opensource.org/faq#cc-zero
[4]https://opensource.org/approval
[5]https://opensource.org/faq

In this study, we investigate to what extent software licenses that do not provide open-source guarantees (or "non-approved licenses" for short) are used in open-source projects published on package managers. Package managers are particularly relevant to license usage due to at least two reasons: (1) they are growing up faster in terms of number of libraries available and packages[6] published [3, 29], and (2) since packages obey a standardized architecture [22], installing and reusing a third-party package comes with no pain. Therefore, packages published in package managers might have a higher number of dependencies than those that do not rely on a package manager. As we shall see in Section 4, on average, a package at NPM has 4.80 dependencies (3rd Quartile: 5, Max: 792).

In this paper we study three well-known package managers: NPM (Node Package Manager), RubyGems and CRAN (The Comprehensive R Archive Network). For each one of these package managers, we downloaded and investigated *all* packages available on them. After this process, we ended up with a comprehensive list of 657,811 software packages scattered through the three well-known, long lived package managers. Specifically, we investigated 510,964 NPM packages, 11,366 CRAN packages, and 135,481 RubyGems packages. Still, in order to provide an evolutionary perspective of the license usage on these packages, we studied 4,367,440 different packages versions (3,539,494 on NPM and 816,580 on RubyGems, and 11,366 on CRAN. We manually analyzed each license employed in each one of these package versions.

This paper makes the following contributions:

(1) We conducted the largest study on licenses usage and evolution targeting ~660k packages (and their 4.3 million versions) published in three well-known package managers (NPM, RubyGems and CRAN).
(2) We studied the impact of the use of non-approved licenses comprehending the whole dependency chain.
(3) We deployed a survey with 76 package publishers (package developers, owners, or authors) to understand how and why do they use non-approved licenses.

2 BACKGROUND ON OPEN-SOURCE LICENSES

The Open Source Definition [17], published by OSI defines 10 properties that a software license must satisfy to be called Open Source. OSI has also established an approval process, through which a license will be approved as Open Source. As of today, only 83 licenses have been approved (although many other have been submitted). Other organizations also approve licenses to be open source, such as the Free Software Foundation (FSF), and the Debian Foundation (these two call them Free Software Licenses—with one exception, the NASA Open Source Agreement 1.3, all OSI approved licenses are considered free software by the FSF[7]).

In the scope of this paper, we consider licenses approved by OSI only. This decision was motivated by the fact that differently than FSF, which can both develop and approve licenses, OSI does not develop — only approves — licenses. Since a license can be submitted by anyone interested in contributing to open-source, the community participation, a crucial aspect of modern open-source software[2, 18], is much more strong at the OSI side.

To better understand the approval process and the implications of not using an OSI approved license, we conducted a semi-structured interview with an OSI's board member. According to him, anybody can submit a license for OSI approval. During the certification process, everyone is invited to participate in the review and discussion about the license. The goal of the certification process is to make sure that the submitted license meets all criteria stated at the Open-Source Definition. If the licenses satisfies the requirements set by the Open-Source definition, the license is approved.

One of the main benefits of using an OSI approved license is the guarantee that OSI—and the open source community at large—has vetted the license and that the license is widely known. Therefore, the community can understand, trust, and use a license. Otherwise, if there was no OSI, everyone could develop a new license and claim that it was open-source; this would require that those using the software hire lawyers to understand such license.

This means that, even if some license is very popular in other domains, such as the Create Commons Zero (CC0) license, software released under CC0 is not open-source software. According to the board member, more importantly, this threat applies recursively: *"if project 'A' (which uses an OSI approved license) depends on project 'B' (which does not use an OSI approved license), this would be as dangerous as project 'A' not using an OSI approved license"*. Nevertheless, if one is interested in publishing software assets only (such as data or images), such open-source data can be safely released under CC0 (the requirements of the OSD does not apply for assets). A similar issue occurs when one does not state any license. In this case, the original author has not granted any rights to the recipient of the software. That is, without a permission from the original author, no one can use, redistribute, or create derivative works. Which is the clearly the opposite of the open-source concepts.

3 METHOD

In this section we present our research questions and method, the data gathered and, our ground definitions.

3.1 Research Questions

The main goal of this study is to gain an in-depth understanding of non-approved open-source licenses. We designed the following three research questions to guide our research:

- **RQ1:** How common are non-approved licenses on software packages?
- **RQ2:** What is the impact of non-approved licenses on the package managers ecosystem?
- **RQ3:** Why developers adopt non-approved licenses?

To answer these questions, we conducted a two-phase research, adopting a sequential mixed-method approach. First, we collected data about license usage and evolution on a corpus of ~660k software packages (Section 3.2). After that, we performed a survey targeting 76 package publishers (Section 3.3).

[6]Throughout this paper, the term "package" refers to any kind of software systems hosted in any of these package managers, regardless of their characteristics.
[7]See https://www.gnu.org/licenses/license-list.en.html

3.2 First study: mining license usage

3.2.1 Package and Package Managers. In our first study, we mined license information of software packages hosted in three well-known, long lived package managers: NPM, RubyGems, and CRAN. The package managers studied have the following characteristics:

- **NPM**[8] manages and indexes Node.js packages. Node.js is a JavaScript runtime environment. The NPM package manager was launched in 2009 and, as of October 2017, it contains over 521K packages. Although it offers support for maintaining packages in-site (it has a version control system), most of the packages available on it are maintained elsewhere (*e.g.,* GitHub). To submit a package to NPM, a user must create an account and push the package using the `NPM` software utility.
- **RubyGems**[9] manages and indexes Ruby packages. RubyGems was launched in 2009 and, as of October 2017, it contains over 192K packages. It also offers support for maintaining packages in-site, but most of the packages published are maintained elsewhere (*e.g.,* GitHub). RubyGems distributes binaries (*i.e.*, a gem file) through its web interface. Anyone interested in submitting a package to RubyGems must create an account and push the package using the `gem` software utility.
- **CRAN**[10] manages and indexes R packages. Differently from NPM and RubyGems, CRAN distributes both the source and binary code of packages published on it. CRAN was launched in 1998 and, as of October 2017, it contains over 11K packages. One interested in submitting a package to CRAN needs to create an account and submit the package through CRAN web interface.

These package managers are the host of several well-known and non-trivial software packages, including React on NPM, Rails on RubyGems, and ggplot2 on CRAN. Packages in these package managers are downloaded millions of times per month. For instance, only on September 2017, on NPM, the packages BlueBird[11], React[12], and Lodash[13] were, in total, downloaded more than 69 million times (18 mi, 6 mi, and 45 mi, respectively). Package managers also make available package releases (*i.e.,* a new version). Table 1 presents the distribution of versions per package. As we can see, the 56% of packages published at NPM have up to three version (58% on RubyGems, and 75% on CRAN). Packages with 10 or more versions are also common (17% on NPM, 16% on RubyGems, but 0.5% on CRAN). Generally speaking, CRAN has less package versions than NPM and RubyGems.

3.2.2 Data Collection. We created an infrastructure to download, extract data, and match dependencies between package versions. Our infrastructure downloaded metadata for *all* packages available on the three package managers. Both NPM and RubyGems provide an API to collect relevant data[14]. Our infrastructure gathers CRAN metadata navigating through its public HTML files. For CRAN and NPM, we collected our data on September the 7th, 2017. We collected RubyGems metadata on September the 15th, 2017. Table 2 depicts the metadata download in each package version for each package manager.

Table 1: Versions Per Package

# of Versions	CRAN	NPM	RubyGems
1	8,848	150,546	42,668
2	1,942	80,243	22,720
3	360	55,028	15,089
4	140	39,890	10,743
5	67	30,192	7,688
6	38	22,886	5,814
7	30	18,190	4,549
8	12	15,105	3,550
9	17	12,000	2,870
$\geq$*10*	67	86,884	19,790

Table 2: Metadata downloaded in each package manager

	CRAN	NPM	RubyGems
Release date	✓	✓	✓
# of Downloads	–	–	✓
# of Licenses	✓	✓	✓
Author	✓	✓	✓
E-mail	✓	✓	–
# of Dependencies	✓	✓	✓
Delimiters	✓	✓	✓

After downloading the metadata, our infrastructure validated whether a (downloaded) package X depends on (also downloaded) a package Y. We validated dependencies using the version number stated in package X and the version number defined in package Y. The three package managers use the notion of delimiters to express a range of possible versions that are compatible with a given package. Example of delimiters include the characters ">", "< ", "*", "~", and "∧". For example, a package X that depends on the 'react' package can declare a dependecy as "`react@^15.0.0`", which indicates that package X depends on any version compatible with `react@15.0.0`. In addition, in the NPM and in the RubyGems, package publishers could use the "x" character to specify a small range of versions (*e.g.,* 1.1.x or 1.x). To match dependencies, we selected the first version available that matched the pattern. As an example, NPM package 'gulp', version '2.6.0' (`gulp@2.6.0` for short) depends on package `event-stream@3.0.x`. As a result, our infrastructure successfully matched package `gulp@2.6.0` to `event-stream@3.0.0` dependency. This match proceedure is important for the impact analysis (**RQ2**).

We downloaded data using three Google Cloud Platform VMs. We used one dual-core VM with 7.5Gb of main memory and 20Gb of SSD, and two single-core VMs with 3.5Gb of main memory and 10Gb of hard disk. After downloading, our dataset occupied 1.2Gb of disk space (1.1Gb of NPM data, 4.6Mb of CRAN data, and 182Mb

[8]https://www.npmjs.com/
[9]https://rubygems.org/
[10]https://cran.r-project.org/
[11]https://www.npmjs.com/package/bluebird
[12]https://www.npmjs.com/package/react
[13]https://www.npmjs.com/package/lodash
[14]For NPM we used https://skimdb.npmjs.com/registry/_all_docs, whereas for RubyGems we used https://rubygems.org/versions

of RubyGems data). The infrastructure used as well as the data collected can be found at the companion website[15].

Table 3 shows the distribution of number of licenses per package version.

Table 3: Licenses Per Version

# of Licenses	CRAN	NPM	RubyGems
0	0	369,914	394,582
1	5,346	3,158,391	419,095
2	5,881	10,287	2,411
3	130	669	355
4	6	222	29
5	1	11	61
6	2	0	46
10	0	0	1

As we can see, the majority of packages have a single license. Interestingly, no package with no license could be found at CRAN. This happens because CRAN does not publish packages without the selection of a license[16]. Still, package versions with two or more licenses are common. For instance, the package `sixarm_ruby_unaccent@1.1.2`, published at RubyGems, was released with 10 licenses (they are: apache-2.0, artistic-2.0, bsd-3-clause, cc-by-nc-sa-4.0, agpl-3.0, gpl-3.0, lgpl-3.0, mit, mpl-2.0, and ruby).

Table 4 presents the number of dependencies per version package. Approximately 29% of NPM package versions have no dependencies (39% for CRAN and 30% for RubyGems, respectively).

Table 4: Dependencies per Version

# of Dependencies	CRAN	NPM	RubyGems
0	6,435	1,047,089	258,810
1	1,782	537,283	194,312
2	1,701	412,121	143,616
3	1,517	322,234	84,679
4	1,183	241,449	51,338
5	978	180,349	31,424
6	733	139,429	22,698
7	521	111,070	13,720
8	436	85,631	11,302
9	323	69,024	8,699
$\geq$*10*	1,060	472,466	32,879

Although the average number of dependencies per package version is 3.8, outliers were found. For instance, the CRAN package `seurat@2.0.1` has 41 dependencies, the RubyGems package `aws-sdk-resources@3.1.0` has 105 dependencies, and the NPM package `primeng-custom@4.0.0-beta.1` has 500 dependencies.

[15]https://github.com/rmeloca/EcosystemsAnalysis

[16]https://cran.r-project.org/web/packages/policies.html

3.2.3 License Groups. As aforementioned, we downloaded metadata for 657,811 software packages (510,964 NPM packages, 11,366 CRAN packages, and 135,481 RubyGems packages), spanning 4,367,440 versions (3,539,494 on NPM and 816,580 on RubyGems, and 11,366 on CRAN). When analyzing the licenses with which each version was released, we found that some of them included typos or wrong names. This happened because NPM and RubyGems allow one to fill the license field with any information. We then manually normalized each license found.

The normalization process was conducted in pairs, followed by conflict resolution meetings. For each license, two authors checked if it (1) was approved by OSI, (2) was not approved but was defined somewhere else, i.e., in the Software Package Data Exchange[17], (3) was not approved neither not defined anywhere else. Licenses not found at OSI list neither at SPDX were allocated in the Other category. To check whether the license was already defined, we searched for its specification on blog posts, Q&A websites, and mailing lists. If the formal specification of a license was not found, the license was included on the non-approved license group. After this process, we ended up with six license groups, namely:

- **OSI licenses:** Any licenses approved by OSI. For this case, we also fixed small issues, such as trivial typos. As an example, we successfully normalized the "apache 2" license to its correct form, "apache-2.0".
- **Incomplete licenses:** Any probably approved license, although we could not fix some issues. For instance, package publishers often omit the version number, *e.g.*, "bsd" or "lgpl", so we could not be sure about which license version was used.
- **SPDX (but not OSI) licenses:** These are the licenses listed in the SPDX License List[18] that were not formally approved by OSI. This group include popular and defined licenses, such as, the "Do What the Fuck You Want to Public License" (WTFPL) or the "Creative Commons Zero" (CC0) license.
- **Missing or Absence of a license**: We aggregated in this group package versions without any license at all (*i.e.,* when package publishers left empty the license field), or developers filled explicit with the NONE word the license field. This is a sub-category of copyright licenses because, as discussed in Section 2, when no license is declared, the original authors retains all rights.
- **Other:** Any licenses with undefined typos, wrong names, or even curses. Examples include: the "d" license and the "Not specified" license. Additionally, we included in this group licenses that the packager publisher put an external link in the license information. We did not inspect each file individually and such data was not included in any of the analysis we conducted because they represent less than 0,5%.
- **Copyright licenses:** This occurs when package publishers explicitly mention that they retain the copyright. Examples include the "my own" license, the "(c) Copyright" license, or the "all rights reserved" license.

At the end of this normalization process, we ended up with 973 distinct licenses (758 at NPM, 46 at CRAN, and 336 at RubyGems).

[17]https://spdx.org/

[18]https://spdx.org/licenses/

Non-approved licenses comprehend all licenses but not OSI licenses and Incomplete licenses.

3.3 Second study: a survey with package publishers

In our second study, we deployed a survey with package publishers of the NPM package manager. We focused on this package manager because (1) the email addresses of the package publishers could be recovered and (2) packages in this package manager exhibits the greatest number of dependencies, which are more likely to affect/be affected, if a license inconsistency is found. We used the following criteria to identify our population: we selected the package publishers of packages versions released under a non-approved license with at least one dependency. This ensures that the irregularity propagates to other packages. After apply the criteria, we obtained 385 package publishers from different project.

Our survey was based on the recommendation Smith *et al.* [21], employing principles for increasing survey participation, such as sending personalized invitations, allowing participants to remain completely anonymous and, asking closed and direct questions as much as possible. Our survey had 14 questions (three of which were open), grouped in three broad interests: demographics (*e.g.,* what is your gender? and what is your profession?), understanding non-approved adoption (*e.g.,* why did you choose? and are you aware of the implications?), and usage frequency (*e.g.,* how often do you use non-approved licenses? and how often do you do not declare a license?). The open questions were analyzed in pairs, followed by conflict resolution meetings. Participation was voluntary and the estimated time to complete each survey was 5-10 minutes. When sending our invitation email, 8 messages were not delivered due to technical reasons. We received 76 responses, representing 20% of response rate. The survey is available at: `https://goo.gl/Jiuwzp`.

4 RESULTS

In this section, we report the results of our study grouped by each research question.

4.1 RQ1. How common are non-approved licenses on software packages?

After the normalization process, we found a total of 973 distinct licenses. These licenses were declared a total of 4,369,024 times. The number of license declarations is higher than the number of its package versions given that one package often employs more than one license (as showed at Table 3). Table 5 shows the distribution of each license group.

As we can see, non-approved licenses (all licenses defined at Section 3.2.3 except OSI licenses and Incomplete licenses) were used 858,311 times, which corresponds to roughly 20% of the overall license usage. Most of them, nevertheless, are related to the absence of a license. We found 764,496 package versions without any license declaration (which is accounts for 89% of the non-approved license usage). In particular, on RubyGems, missing licenses correspond to 48% of the total license used (10.41% on NPM).

We also studied license usage through an evolutionary perspective. In order to provide a general overview, Table 6 groups evolution

Table 5: License Groups on Package Versions

Group	CRAN	NPM	RubyGems	TOTAL
OSI	15,724	3,009,782	403,693	3,429,199
INCOMPLETE	34	73,647	7,833	81,514
SPDX but not OSI	162	30,688	6,215	37,065
MISSING	8	400,618	396,178	796,804
OTHER	220	10,978	4,953	16,151
COPYRIGHT	0	7,106	1,185	8,291

patterns of license changes. We pairwise analyzed all versions available in order to verify how many times a license changed from one group to another. The results show that package versions, regardless of the package manager, tend to propagate their license used over their versions. Therefore, the main diagonal always have the higher values. For instance, at NPM we found that 311,455 package versions without any license associated still had this non-approved license in the next version.

Table 6: Patterns of license evolution

NPM						
From\To	*OSI*	*INC*	*SPDX*	*MISS*	*OTH*	*COP*
OSI	2,576,692	3,012	2,060	2,125	423	116
INC	4,573	61,535	44	144	363	205
SPDX	2,153	26	25,489	182	78	56
MISS	8,911	321	256	337,711	87	23
OTH	502	345	99	51	9,231	241
COP	200	212	58	19	267	6,424
RubyGems						
From\To	*OSI*	*INC*	*SPDX*	*MISS*	*OTH*	*COP*
OSI	336,639	505	574	380	553	37
INC	854	6,575	99	5	116	0
SPDX	618	82	5,095	51	270	1
MISS	8,112	329	279	324,153	185	10
OTH	808	119	272	9	4,197	14
COP	50	1	1	5	15	1,029

Since the changes from approved to non-approved are the most relevant ones to our study, we counted how many times a package version changed from an OSI-approved license to a non-approved license, and vice-versa. We identified these changes in 12,491 packages at RubyGems and 24,075 packages at NPM. Among these package, on RubyGems, 10,442 package versions changed from a non-approved to an approved license. In this case, the publishers corrected their wrong license as presented in Table 8.

Interestingly, the number of changes from an approved to a non-approved licence was much lesser. On RubyGems, we found only 2,049 package versions that changed from an approved license to a non-approved license. A similar behavior occurred at NPM. The number of changes from a non-approved license is much greater than the opposite: (16,339 package versions changed from a non-approved license to an approved one, whereas 7,736 package versions changed from an approved to a non-approved one). As an example, when upgrading from `zorg@0.0.1` to `zorg@0.0.10`, the NPM package changed from the know "ISC" license to no license at all. We did not performed this analysis at CRAN because it does not provide such information.

To provide a more fine-grained perspective about the evolution patterns, we analyzed the top 10 most common changes from an approved license to a non-approved license, and vice-versa. Table 7 presents the evolution patterns, focusing on changes from an approved to a non-approved license. The majority of changes observed were when changing from MIT license to no license at all (1,286 instances found on NPM, and 248 on RubyGems). The effects of a missing license are exactly the opposite a developer might think: it applies the copyright instead of opening the source code. Therefore, the migration from a missing license to the MIT license can be explained as a correction of this effect, specially due to the permissive characteristics of such license. This evidence is supported by Almeida [1] and by ours findings that developers might not fully understand the licensing process of a software.

Table 7: The 10 Most Common License Evolution Patterns: From Approved to Non-Approved

NPM		RubyGems	
Evolution Patterns	#	*Evolution Patterns*	#
mit → missing	1,286	mit → missing	248
isc → missing	604	apache-2.0 → missing	85
apache-2.0 → missing	116	bsd-3-clause → missing	33
bsd-2-clause → missing	37	lgpl-2.0 → missing	4
gpl-3.0 → missing	20	gpl-3.0 → missing	4
bsd-3-clause → missing	19	bsd-2-clause → missing	2
gpl-2.0 → missing	12	gpl-2.0 → missing	2
lgpl-3.0 → missing	9	lgpl-3.0 → missing	1
fair → missing	9	ms-pl → missing	1
mpl-2.0 → missing	7	–	–

Table 8 presents the evolution patterns between licenses, but now focusing in changes from a non-approved to an approved license. In RubyGems, we found that the majority of the cases changed from a missing license to MIT (6,556 instances), Apache-2.0 (614 instances), GPL-3.0 (239 instances), or GPL-2.0 (153 instances). A similar pattern occurs on NPM. Most of the changes are from a missing license to MIT (6,667), ISC (831 instances), Apache-2.0 (633 instances), or BSD-3-CLAUSE (262 instances).

RQ1 Summary. We found 1,058,554 packages versions (24.23%) released under non-approved licenses. Packages published on RubyGems are the most affected ones (55% of them employed a non-approved license). The missing (lack of a license) license is widespread. When license change occurs, most of the package versions keep the same license, although changes from a non-approved to an approved license, and vice-versa, are common.

Table 8: The 10 Most Common License Evolution Patterns: From Non-Approved to Approved

NPM		RubyGems	
Evolution Patterns	#	*Evolution Patterns*	#
missing → mit	6,667	missing → mit	6,556
missing → isc	831	missing → apache-2.0	614
missing → apache-2.0	633	missing → gpl-3.0	239
missing → bsd-3-clause	262	missing → gpl-2.0	153
missing → gpl-3.0	137	missing → bsd-3-clause	133
missing → bsd-2-clause	91	missing → lgpl-3.0	86
missing → gpl-2.0	85	missing → bsd-2-clause	81
missing → lgpl-3.0	61	missing → artistic-2.0	73
missing → mpl-2.0	49	missing → agpl-3.0	33
missing → agpl-3.0	35	missing → lgpl-2.1	31

4.2 RQ2. What is the impact of non-approved licenses on the package managers ecosystem?

To understand the impact of a non-approved license, we calculated two types of metrics (irregular and affected) in three different granularities (graph order).

- **Irregular**. A package is called irregular if at least one of its versions has a direct dependency to a package released under a non-approved license. If a package is irregular it means that it can affect other packages that depends on it.
- **Affected**. A package is affected if at least one of its versions has direct or indirect dependency to a package that is irregular. Direct dependency is when one package father (affected) depends on its child (irregular). Indirect dependency is when there are more than one level between affect and irregular packages.

With these metrics, we analyzed the whole dependency graph of all package versions. Table 9 shows the impact of non-approved licenses in terms of packages, versions, and dependencies. In terms of packages, although NPM have more irregular and affected packages, RubyGems presents a higher proportion of irregular (46% vs 18%) and affected (55% vs 38%) packages than NPM, which suggests that almost half of all package versions on RubyGems are irregular. The low number of packages, versions, and dependencies affected at CRAN is because CRAN prevents the absence of licenses by requiring package publishers to choose at least one from their license selection. Again, when we projected the impact including the indirect dependencies of each package version, the impact in NPM is higher than RubyGems, because NPM packages have more versions.

To provide a more detailed example, Figure 1 shows a fragment of a dependency graph of the package `request@2.81.0`. This particular package has 23,205 direct dependencies to it (6,840 are irregular) and 42,938 indirect dependencies to it (parents). Moreover, we omitted from Figure 1 the regular direct dependencies. In the figure, solid lines edges are regular dependencies and dotted lines edges are irregular dependencies. Double border lines vertexes are regular package versions whereas single solid border ones are irregular.

Table 9: Impact caused by non-approved licenses in each package manager

Graph Order	*Metric*	***CRAN***	***NPM***	***RubyGems***
	#	11,366	510,964	135,481
	Irregular	1082	78,224	62,967
Packages	***Proportion***	0.095	0.153	0.464
	Affected	1455	194,741	75,475
	Proportion	0.128	0.381	0.557
	#	11,366	3,539,494	816,580
	Irregular	35	690,703	440,443
Versions	***Proportion***	0.003	0.195	0.539
	Affected	36	1,619,248	520,967
	Proportion	0.003	0.457	0.637
	#	1,086	15,521,508	1,765,288
Dependencies	*Irregular*	59	1,364,281	1,088,298
	Proportion	0.054	0.087	0.616

Dotted border vertexes represents affected packages. Notice that a package might be irregular and affect at the same time.

We also observed that in this fragment of the graph, three packages have a non-approved missing license associated with: "assert-plus", "verror", and "extsprintf". It is worth to mention that package "assert-plus" and "extsprintf" are considered regular packages because they do not have a dependency to any package version released under a non-approved license.

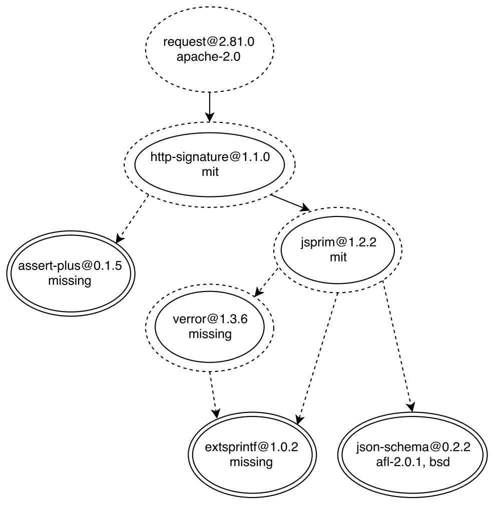

Figure 1: Example of a affected package version dependency tree

Another example occurs on RubyGems package manager: the package `activesupport`, actually on version 4.2.6, was downloaded 174,538,434 times on its entire life cycle, but in the version 4.0.0, released on 2013 (25th June), this package was depending to the unlicensed packages minitest@4.2.0, multi_json@1.3.3, thread_safe@0.1.0 and tzinfo@0.3.37 (activesupport also was dependending to the MIT-licensed package i18n@0.6.4). This particular version was downloaded 3,107,216 times and was used by 1,093 another published packages directly and by 16,526 packages taking into account both direct and indirect dependencies. The package activesupport is a toolkit extracted from the Rails framework's core.

To provide an extra perspective of the impact of non-approved licenses, we compared the number of irregular and affected values with incomplete licenses. We chose incomplete licenses because they can be interpreted as wrong licenses, since they do not have a correct name or version license.

Table 10 presents the most common incomplete licenses per package manager. Among the most incomplete licenses, we observed that package publishers are using a number of licenses omitting its version.

Table 10: Top 10 Incomplete Licenses

CRAN		NPM		RubyGems	
License	#	License	#	License	#
agpl	12	bsd	59,132	bsd	4,280
bsd	11	gpl	7,904	gpl	1,783
cecill	6	lgpl	2,747	lgpl	1,067
mpl	2	epl	1,173	agpl	304
epl	2	mpl	854	artistic	166
bsl	1	agpl	832	epl	71
—	—	free	218	mpl	50
—	—	ibm	216	free	36
—	—	apl	194	osl	26
—	—	cecill	179	afl	16

In this sense, Table 11 presents the impact of Incomplete licenses. It is worth to mention that even if we consider the incomplete licenses as inconsistent licenses, non-approved licenses (9) presented a higher impact than Incomplete licenses, for instance, the number of irregular packages caused by non-approved licenses are 62,154 against 63,329 irregular packages caused by Incomplete licenses on RubyGems (the ratio of the difference 813/362 is almost 2.5 times higher). If we compare the affected versions on RubyGems, the impact of non-approved licenses are almost 69 times higher than the Incomplete Licenses. In a general way, we also found that NPM is more affected by Incomplete licenses than RubyGems.

Finally, CRAN packages were highly impacted by Incomplete licenses, which is mostly due to the lack of a license version. This behavior turns ~11% of CRAN packages irregulars, which affects almost 15% of the published packages.

We recognize that non-approved licenses are dangerous to both package authors (publishers on package managers) and users – that create but not explicit publish a package with direct dependencies to published packages – because of the uncertainty whether the dependencies of the desired-to-publish package are regular or not. In fact, package publishers should look at the whole dependency chain. However, a few factors might imply in the presence of such irregularities in package managers, such as the height of the package dependency tree, and the presence of newcomers at

Table 11: Impact caused by Incomplete licenses in each package manager

Graph Order	*Metric*	***CRAN***	***NPM***	***RubyGems***
	#	11,366	510,964	135,481
	Irregular	1,256	94,515	63,329
Packages	***Proportion***	0.110	0.184	0.467
	Affected	1,480	197,626	75,455
	Proportion	0.130	0.386	0.556
	#	11,366	3,539,494	816,580
	Irregular	38	825,520	443,072
Versions	***Proportion***	0.003	0.233	0.542
	Affected	38	1,639,430	520,836
	Proportion	0.003	0.463	0.637
	#	1,086	15,521,508	1,765,288
Dependencies	*Irregular*	62	1,759,643	1,098,489
	Proportion	0.057	0.113	0.622

the open source community, who might not be completely aware about license constraints.

RQ2 Summary. Non-approved licenses impact packages from NPM and RubyGems, making packages irregular and affecting both its direct and indirect dependencies. Non-approved licenses can be considered more harmful than incomplete licenses since their impact is higher when compared to the amount of irregular and affected packages and versions by each License group.

4.3 RQ3. Why developers adopt non-approved licenses?

To answer this question, we report the results of our survey with 76 package publishers. Our target population is 94% male and 96% work for the software development industry. About 53% of them have created or contribute to up to 30 open-source projects (18% of them have created or contribute to more than 100 open-source projects). Still, 48% of the respondents believe that about 20% of these created/contributed open-source projects use a non-approved license. More interestingly, however, is the fact that 27% of the respondents have no idea about how many projects they contribute use a non-approved license. Similarly, in Section 4.1, we showed evidence that about 18% of the package versions studied use a non-approved license.

When we asked why do they use a non-approved license, we found that 26 of the respondents **do not care** about the specific license terms. Along this line, one respondent mentioned that "*I choosed WTFPL license because I really don't care about who and how use my modules. I share my code with people and it's a pleasure for me to just know if someone finds it useful. May be if I wrote something really great like Facebook's React I would think about fame*". Also, 17 respondents acknowledged that using a non-approved license was a **naive decision**: "*I thought I was appropriate*". Still, **small projects** seem to be more prone to be licensed under a non-approved license. Yet, 5 respondents are aware that a non-approved license makes sense when **licensing non-software projects**, for instance, "*Because it fits the content of the repository best (it is not a source code repository, but contains only data)*". Finally, some developers adopt non-approved licenses because they claim they are **simpler** (6 occurrences) or **more open** (4 occurrences), for instance, one respondent said that she likes "*the idea of WTFPL. Makes everything pretty clear. You just do what you want..*

Right afterwards, we asked whether they are aware of the implications of using a non-approved license; 43% of the respondents mentioned a lack of awareness. For those who mentioned to be aware of the implications, we asked them to cite one example of an implication. Among the answers we found that developers believe that a non-approved license might **limit the adoption** of their software (12 occurrences). As an example, one respondent said that "*If you use a license others have never heard of, others are less likely to contribute and/or may be wary of using you software.*" **Code thefts** was also a recurring implication, mentioned by 7 respondents. Finally, one respondent raised the fact that the main implication of using a non-approved license is that "*it can't be automatically recognized by machines to categorize software under any license which may exclude the software from search results*". This is particularly interesting, since Github helps the project owners to choose a correct license for their repositories. However, the Github help documentation also highlight to developers that they are responsible to define the correct license as we can see on this paragraph: "GitHub provides the information on an as-is basis and makes no warranties regarding any information or licenses provided on or through it, and disclaims liability for damages resulting from using the license information."

In the next five following questions, we asked how often do they (Q9) investigate if the license that you chose conforms with the license that your project depends, (Q10) do not declare a license, (Q11) use a non-approved license, (Q12) use a copyright license in one open-source software, and (Q13) use more than one license (either approved or not)? Figure 2 shows the results.

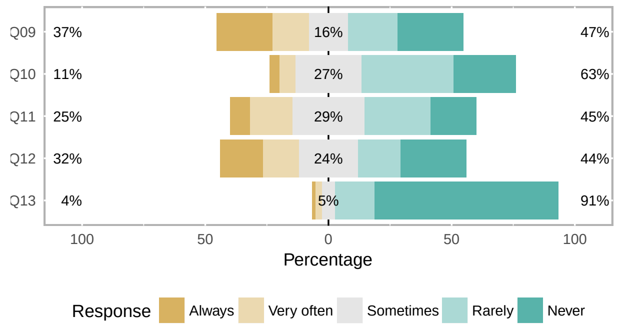

Figure 2: How often do you (Q9) investigate if the license that you chose conforms with the license that your project depends, (Q10) do not declare a license, (Q11) use a non-approved license, (Q12) use a copyright license in one open-source software, and (Q13) use more than one license (either approved or not)?

This figure shows a couple of interesting information. First, we can see 46% of respondents "Never" or "Rarely" take into account the license used in the software's dependencies. We believe this is an important result because, as we discussed in Section 2, licenses inconsistencies directly impact any project that depends upon. With similar implications, 11% of the respondents "Always" or "Very Often" do not declare a license. One respondent even mentioned that she "*Frequently forget to declare any license and it seems unimportant.*" Similarly, 25% of the respondents "Always" or "Very Often" use a non-approved license. Finally, 94% mentioned that they "Never" or "Rarely" use more than one license (either approved or not). One respondent mentioned that the reasons of why she uses more than one license is related to the fork-based model: "*TypoPRO is a collection(!) of fonts and each font already has its distinct Open Source software license from their upstream vendor. So, TypoPRO stays under (this union) set of upstream licenses.*"

RQ3 Summary. 26 respondents do not care about the license used. Some respondents believe that non-approved licenses are more open and simpler to use. Among the implications, 12 respondents believe that non-approved licenses can limit the adoption of their software. 46% of the respondents do not take license into account when choosing a package dependency.

5 IMPLICATIONS

This research has implications for different kinds of stakeholders. Three of these possible groups are discussed below.

Package managers. Since we observed that both NPM and RubyGems do not require developers to inform a license, many packages published on these packages managers either (1) do not use any license or (2) state a wrong or incomplete license name (**RQ1**). This problem not only hinders researchers from conducting in-depth studies on license usage, but also have the potential of confusing software developers interested in using the software package. Package managers, therefore, might introduce mechanisms to prevent the introduction of wrong (or even non existing) license names.

Researchers. Although software licensing is an established research topic, our notion of non-approved licenses was not yet fully explored (**RQ1**) and its implications were unclear (**RQ2**). Researchers can expand our comprehension of non-approved licenses in many ways. First, researchers could introduce mechanisms to automatically detect the use of non-approved licenses. Still, since packages tend to propagate their licenses over the releases (**RQ1**), researchers can create techniques to avoid non-approved license propagation.

CS Professors. Educators can also benefit from the findings of this study. Since software license is a common misunderstood topic among software developers [1], software engineering professors could bring problems related to license usage to the classroom, and invite students to discuss possible solutions or compare to the perception of professional software developers (**RQ3**). Similarly, in order to make software licenses more appealing to aspiring software engineers, professors can use our license inconsistency graph (**RQ2**) in advanced data-structure classes, and invite students to understand license inconsistencies in complex and deeper graphs.

6 THREATS TO VALIDITY

In a study of such proportion, there are always many limitations and threats to validity. First, we could not retrieve data from 2,140 packages (1,079 NPM packages, 1,052 RubyGems packages, and 9 CRAN packages). This happened because such packages metadata could not be located. However, these packages represent only 0.04% of the whole universe of packages from our study.

Second, the normalization process was manual and, therefore, error-prone. We mitigated this threat using pair-review work. Each author independently analyzed the same set of licenses, with subsequent conflict resolution meetings. Both the original and normalized license sets are available for future analysis. We choose not to analyze the external FILE licenses because most of these package versions are hosted on GitHub and would require manual search for the license file into the repositories. At CRAN 1,391 package versions have a file license declared; on NPM, 19,010; and, RubyGems have more than 20,000 package version using the FILE license.

Third, one might argue that our packages studied might be full of simple, trivial software projects. However, packages available on package managers are often more mature, when compared to software projects hosted on other coding websites such as Github, which are often personal projects and class projects [9].

Fourth, we rely on the licenses approved by OSI. Even if a license is commonplace — for instance, we found 4,927 package versions using the creative commons zero (CC0) license (104 at CRAN, 3,022 at NPM and 1,801 at RubyGems) — we still consider such licenses as non-approved. Although we are aware that many other institutions such as the Free Software Foundation (FSF) and the Debian Foundation approve licenses, we decided to stick to OSI approval because: (1) licenses can be submitted by anyone interested in to get an OSI approve, and (2) licenses approved by OSI are commonly used — as we shown in Table 5, there are only few licenses found in our dataset that were not recognized by OSI.

Finally, we did not double checked whether the license informed at the package manager was, indeed, the same declared at the official package website. We chose not to validate the license used due to two reasons: first, the package publisher (which is often a core member of the project) is in charge of declaring the license used in a given published version. That is, no one other than the package publisher would be more confident to state the correct license used; second, because we manually studied hundreds of thousands of software packages. These software packages are often hosted in a third-party coding website (*e.g.,* GitHub or BitBucket), which store license information using distinct ways, *e.g.:* Github shows the license name at the project's first page (if the algorithm succeed at inferring the license, which is not always the case); BitBucket, on the other hand, does not explicitly demand any license when creating a repository. Additionally, if the project has the proper license file, it will display the license on the project's cover page. This problem only exacerbates when considering license information per version release. Therefore, due to the lack of standards and our substantial sample size, performing such manual process would be prohibitive.

7 RELATED WORK

Recent studies investigated licenses inconsistencies, which is a similar to our concept of non-approved licenses. Since non-approved licenses also introduce inconsistencies, one can see non-approved as a subset of license inconsistencies. However, we believe that the implication of non-approved licenses are greater than the known problems related to licenses inconsistencies.

To the best of our knowledge, our work is the first to analyze the usage and adoption of Non-Approved licenses. We also discussed the impact of Non-Approved licenses compared to incomplete licenses in the package manager context, which have attracted more attention from practitioners and researchers, since NPM, CRAN and RubyGems are growing faster and becoming increasingly popular. We summarize the related work in terms of *licenses maintenance and evolution* and *licenses inconsistencies.*

Di Penta *et al.* [4] proposed a method to track the evolution of software licensing and investigated its relevance on six open source projects. Most of the inconsistencies found were related to files without a license. Vendome *et al.* [24, 27] conducted a large empirical study investigating when and why developers adopt or change software licenses. Recently, Vendome *et al.* [26] performed another large-scale empirical study on the change history of over 51K FOSS systems to investigate the prevalence of known license exceptions, presenting a categorization and a Machine Learning-Based Detection algorithm to do identify license exceptions. Santos [20] analyzed a set of 756 projects from FLOSSmole repository of Sourceforge.net data that had changed their source code distribution allowances. The author found 88 projects with a "none" license – which might leave projects exposed and legally unattended – and, 55 times where projects changed their current state of having a license to one where they have no license.

German *et al.* [8] investigated how the licenses declared in packages are consistent with the source code files in the Fedora ecosystem. Manabe *et al.* [15] extended it by proposing a graph visualization to understand those relationships. They found that the GPL Licenses are more likely to include other licenses, while Apache Licenses tend to contain files only under the same license. The authors reported changes from a valid license to none and some cases where a non-valid license was changed to a valid license.

Wu *et al.* [31, 32] investigated license inconsistencies caused by re-distributors that removed or modified the license header in the source code. The authors described and categorized different types of license inconsistencies, proposing a method to detect them in the Debian ecosystem. The authors found that, on average, more than 24% of packages relationship have a "none" license between them, however this effect was not discussed. Wu *et al.* [30] also studied whether the issues of license inconsistencies are properly solved by analyzing two versions of Debian investigating the evolution patterns of license inconsistencies, which will disappear when the downstream projects get synchronized.

Lee *et al.* [14] compared machine-based algorithms to identify potential license violations and guide non-experts to manually inspect violations. The authors reported that the accuracy of crowds is comparable to that of experts and to the machine learning algorithm. Interesting to note that approximately 25% of files from 227 projects (79.4 % of projects analyzed) did not have any license.

Almeida *et al.* [1] conducted a survey with 375 developers to understand whether they understand violations and assumptions from three popular open source licenses (GNU GPL3.0, GNU LGPL 3.0 and MPL 2.0) both alone and in combination. The authors confront the answers with expert's opinion, and found that the answers were consistent in 62% of 42 cases. Although previous work in understanding software licenses pointed "None" as frequently choose for files and packagers, neither scenario involved this aspect.

Van der Burg *et al.* [23] proposed an approach to construct and analyze the Concrete Build Dependency Graph (CBDG) of a software system by tracing system calls at build-time. Through a case study of seven open source systems, the authors showed that the constructed CBDGs can accurately classify sources as included in or excluded from deliverables with 88%-100% precision and 98%-100% recall, and can uncover license compliance inconsistencies in real software systems. German and Di Penta [6] presented a method for open source license compliance of Java applications. The authors implemented a tool called Kenen, to mitigate any potential legal risk for developers that reuse open source components. Kapitsaki *et al.* [11] compared tools that are used to detect licenses of software components and avoid license violations, classifying them in three types: License information identification from source code and binaries, software metadata stored in code repositories, and license modeling and associated reasoning actions.

8 CONCLUSION

In this paper we conducted a large-scale study on non-approved licenses, in terms of usage, impact, and adoption. Non-approved licenses are any license not approved by OSI, the Open Source Initiative. Software released under a non-approved license cannot be claimed to be open-source (the original author retains all rights). Non-approved licenses include licenses with typos, wrong names, or even curses, or even missing licenses (e.g., when package publishers do not fill the license information).

When mining data from ~657k open-source projects, we observed that hundreds of non-approved licenses exist. About 24% of the packages released used at least one of these non-approved licenses. The majority of non-approved licenses found are, in fact, the absence of a license. Still, we found that package publishers tend to propagate the same license used though package versions. Non-approved licenses impact packages from NPM and RubyGems more than Incomplete licenses when we compared to the amount of irregular and affected packages and versions. Finally, when we asked packagers publishers about non-approved license, we found that 46% of the respondents do not take license into account when choosing a package dependency, some respondents believe that non-approved licenses are more open and simpler to use. On the other hand, 12 respondents believe that non-approved licenses may limit the adoption of their software.

For future work, we plan to investigate the evolution of non-approved licenses in a fine-grained way (*e.g.,* through commits instead of version releases). This would deepen our understanding on why non-approved licenses are adopted. Still, since CRAN developers might have a more diverse background (*e.g.,* biologists, mathematicians, among others), we plan to get in touch with them to understand their motivations behind the usage of non-approved licenses.

ACKNOWLEDGMENTS

This work is supported by Fundação Araucária; CNPq (406308/2016-0 and 430642/2016-4); PROPESP/UFPA; and FAPESP (2015/24527-3).

REFERENCES

[1] 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. https://doi.org/10.1109/ICPC.2017.7

[2] Jailton Coelho and Marco Tulio Valente. 2017. Why Modern Open Source Projects Fail. In *25th International Symposium on the Foundations of Software Engineering (FSE)*. 186–196.

[3] Eleni Constantinou and Tom Mens. 2017. An Empirical Comparison of Developer Retention in the RubyGems and Npm Software Ecosystems. *Innov. Syst. Softw. Eng.* 13, 2-3 (Sept. 2017), 101–115. https://doi.org/10.1007/s11334-017-0303-4

[4] Massimiliano Di Penta, Daniel M. German, 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 '10)*. ACM, New York, NY, USA, 145–154. https://doi.org/10.1145/1806799.1806824

[5] Karl Fogel. 2017. *Producing Open Source Software: How to Run a Successful Free Software Project* (second ed.). O'Reilly Media. http://www.producingoss.com/.

[6] D. German and M. Di Penta. 2012. A Method for Open Source License Compliance of Java Applications. *IEEE Software* 29, 3 (May 2012), 58–63. https://doi.org/10.1109/MS.2012.50

[7] Daniel M. German and Jesús M. González-Barahona. 2009. *An Empirical Study of the Reuse of Software Licensed under the GNU General Public License.* Springer Berlin Heidelberg, Berlin, Heidelberg, 185–198. https://doi.org/10.1007/978-3-642-02032-2_17

[8] D. M. German, M. Di Penta, and J. Davies. 2010. Understanding and Auditing the Licensing of Open Source Software Distributions. In *2010 IEEE 18th International Conference on Program Comprehension.* 84–93. https://doi.org/10.1109/ICPC.2010.48

[9] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, DanielM. German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

[10] 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.

[11] Georgia M. Kapitsaki, Nikolaos D. Tselikas, and Ioannis E. Foukarakis. 2015. An insight into license tools for open source software systems. *Journal of Systems and Software* 102 (2015), 72 – 87. https://doi.org/10.1016/j.jss.2014.12.050

[12] Cory Kapser and Michael W. Godfrey. 2008. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering* 13, 6 (2008), 645–692.

[13] Miryung Kim, L. Bergman, T. Lau, and D. Notkin. 2004. An ethnographic study of copy and paste programming practices in OOPL. In *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on.* 83–92.

[14] Sanghoon Lee, Daniel M German, Seung-won Hwang, and Sunghun Kim. 2015. Crowdsourcing Identification of License Violations. *Journal of Computing Science and Engineering* 9, 4 (2015), 190–203.

[15] Yuki Manabe, Daniel M. German, and Katsuro Inoue. 2014. *Analyzing the Relationship between the License of Packages and Their Files in Free and Open Source Software.* Springer Berlin Heidelberg, Berlin, Heidelberg, 51–60. https://doi.org/10.1007/978-3-642-55128-4_6

[16] Trevor Maryka, Daniel M. German, and Germán Poo-Caamaño. 2015. *On the Variability of the BSD and MIT Licenses.* Springer International Publishing, Cham, 146–156. https://doi.org/10.1007/978-3-319-17837-0_14

[17] OSD. 2018. The Open Source Definition (Annotated). (2018). https://opensource.org/osd-annotated

[18] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. 2016. More Common Than You Think: An In-depth Study of Casual Contributors. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1.* 112–123.

[19] Lawrence Rosen. 2004. *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

[20] Carlos Denner dos Santos. 2017. Changes in free and open source software licenses: managerial interventions and variations on project attractiveness. *Journal of Internet Services and Applications* 8, 1 (07 Aug 2017), 11. https://doi.org/10.1186/s13174-017-0062-3

[21] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann. 2013. Improving developer participation rates in surveys. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 89–92. https://doi.org/10.1109/CHASE.2013.6614738

[22] Diomidis Spinellis. 2012. Package Management Systems. *IEEE Software* 29, 2 (2012), 84–86.

[23] Sander van der Burg, Eelco Dolstra, Shane McIntosh, Julius Davies, Daniel M. German, and Armijn Hemel. 2014. Tracing Software Build Processes to Uncover License Compliance Inconsistencies. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 731–742. https://doi.org/10.1145/2642937.2643013

[24] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. 2017. License usage and changes: a large-scale study on gitHub. *Empirical Software Engineering* 22, 3 (01 Jun 2017), 1537–1577. https://doi.org/10.1007/s10664-016-9438-4

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

[26] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. 2017. Machine Learning-based Detection of Open Source License Exceptions. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 118–129. https://doi.org/10.1109/ICSE.2017.19

[27] Christopher Vendome, Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. 2015. When and Why Developers Adopt and Change Software Licenses. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) (ICSME '15)*. IEEE Computer Society, Washington, DC, USA, 31–40. https://doi.org/10.1109/ICSM.2015.7332449

[28] Christopher Vendome, Mario Linares-Vasquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. 2015. When and Why Developers Adopt and Change Software Licenses. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) (ICSME '15)*. IEEE Computer Society, Washington, DC, USA, 31–40. https://doi.org/10.1109/ICSM.2015.7332449

[29] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. 2016. A Look at the Dynamics of the JavaScript Package Ecosystem. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 351–361. https://doi.org/10.1145/2901739.2901743

[30] Yuhao Wu, Yuki Manabe, Daniel M. German, and Katsuro Inoue. 2017. *How are Developers Treating License Inconsistency Issues? A Case Study on License Inconsistency Evolution in FOSS Projects.* Springer International Publishing, Cham, 69–79. https://doi.org/10.1007/978-3-319-57735-7_8

[31] Y. Wu, Y. Manabe, T. Kanda, D. M. German, and K. Inoue. 2015. A Method to Detect License Inconsistencies in Large-Scale Open Source Projects. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories.* 324–333. https://doi.org/10.1109/MSR.2015.37

[32] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. German, and Katsuro Inoue. 2017. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering* 22, 3 (01 Jun 2017), 1194–1222. https://doi.org/10.1007/s10664-016-9487-8