Journal of Information Processing Vol.29 296–304 (Apr. 2021)

[DOI: 10.2197/ipsjjip.29.296]

Regular Paper

Empirical Study on Dependency-related License Violation in the JavaScript Package Ecosystem

Shi Qiu[1,a) Daniel M. German[2,b) Katsuro Inoue[1,c)

Received: August 3, 2020, Accepted: January 12, 2021

Abstract: Open source software (OSS) is software whose source code can be reused under some particular terms and conditions. These terms and conditions are usually described by one or more software licenses written in the header part of the source files. A license may violate another one according to the terms and conditions. Making software by reusing OSS as dependency may cause dependency-related license violation if the developers overlook the license of the dependency. In this paper, we first conduct an empirical study on npm - a JavaScript-based software ecosystem - to study the prevalence of dependency-related license violation. The result suggests that only a few packages (0.644%) in npm have dependency-related license violations. However, we also observe that including the packages licensed under copyleft licenses in the dependency network potentially causes a high dependency-related license violation. We then conduct a preliminary questionnaire on the authors of packages detected as having dependency-related license violations to study the developers' attitudes. The results reveal: 1) the developers' overlooking and misunderstanding of the dependency-related license violations; 2) the difficulties in managing dependency-related license violations and the developers' demands for help.

Keywords: software maintenance, open source software, software license, OSS ecosystem, license violation

1. Introduction

Software reuse has long been proved to be a good method to increase software productivity [3], [15], [17]. As a practice, reusing open source software (OSS) has become more and more popular.

Open source license describes the terms and conditions when OSS is used, modified, and shared. OSS should be distributed under one or multiple open source licenses so that it can be reused by others. These licenses are usually included in the header comments of source files. To standardize the use of open source licenses, the Open Source Initiative (OSI) determines the definition of open source licenses and publishes the list of all approved licenses [*1]. When developers reuse OSS, they should pay special attention to open source licenses to prevent potential legal risks [25].

As the definition of open source license describes, OSS can only be reused as long as the particular terms and conditions are satisfied. Therefore, the developed software should satisfy all the terms and conditions in licenses of all reused OSS. In other words, the developed software should select a license that does not violate the licenses of all reused OSS. In this paper, we define *license A violates license B* as the situation that the software licensed under license B cannot be combined into the software licensed under license A according to their terms and conditions. If the selected license violates any license of the reused OSS, potential legal risks may occur.

An OSS ecosystem consists of software projects that are developed and evolve together in a shared environment [14]. The user-contributed OSS ecosystem is an ecosystem where software projects are contributed by its users. When a developer develops a new software project, other software projects in the user-contributed OSS ecosystem can be reused easily by being declared as dependencies. In this paper, a *dependency* of a package refers to the other packages in the user-contributed OSS ecosystem used by this package. Meanwhile, this package is defined as a *dependent*. Generally, the dependency can also refer to the dependency relation between the packages [4], but note that in this paper we define dependency as the "package" instead of the "relation". Introducing dependencies makes a project dependent on them.

The license of the new software project under development should not violate the licenses of any dependencies as well. Dependency-related license violation occurs when the selected license violates any license of the dependencies. In a user-contributed OSS ecosystem, it is more difficult to judge whether the selected license violates the licenses of its dependencies or not.

To address the dependency-related license violation issue, in this paper we propose an approach to detect dependency-related license violations of software projects in such OSS ecosystems. We are interested in studying the prevalence of dependency-related license violation in user-contributed OSS ecosystems and

[1] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[2] Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada
[a)] qiujitsu@ist.osaka-u.ac.jp
[b)] dmg@uvic.ca
[c)] inoue@ist.osaka-u.ac.jp

[*1] https://opensource.org/licenses/alphabetical

the developers' attitudes.

We select npm [*2] as the target in our research. npm serves as a large repository of JavaScript-based software packages. It hosts over 1.3 million JavaScript packages and becomes the largest JavaScript ecosystem, with millions of packages being installed from the npm repository on an everyday basis. We select npm for 3 reasons: (1) npm is one of the most popular and successful OSS ecosystems and hosts a large number of packages. (2) npm has a perfect mechanism of including other packages as dependencies, which make the usage of dependencies prevalent in npm. (3) npm has a strict requirement of adding meta-data, for which we can utilize the meta-file of packages conveniently. These features make npm a suitable target to study dependency-related license violations. Studying dependency-related license violations in such typical OSS ecosystems will benefit the practitioners. The findings will also be a good baseline to study dependency-related license violations in other OSS ecosystems.

Our research questions are set as follows:

RQ1: How prevalent are dependency-related license violations in npm?

RQ2: What are the developers' attitude towards dependency-related license violation?

Many studies in software engineering have been done on the software license. Some effective approaches and tools are proposed to identify the license of source code files automatically [7], [10]. Some works aim to detect and understand the license violation in code siblings or similar files [9], [24], but no research has been done to understand the license violation occurring in the dependencies of packages in the OSS ecosystem.

The contributions of this study are as follows.

(1) An empirical study on npm to understand the prevalence of dependency-related license violation with the proposed method of detecting dependency-related license violation.

(2) A preliminary questionnaire on the authors of packages detected as having dependency-related license violation to reveal the developers' attitudes.

This paper is organized as follows. Section 2 first provides a brief background on dependency-related license violation. Our empirical study on npm is described in Section 3, followed by Section 4 with the preliminary questionnaire. Section 5 describes threats to validity. After a description of related work in Section 6, Section 7 concludes this paper.

2. Background

2.1 License Violation

A software license permits software to be reused under certain terms and conditions. An open source license is a software license that follows Open Source Definition [*3] and is approved by Open Source Initiative. Here is an example of a license statement abstracted from grunt package in npm, which states that the file is licensed under MIT license:

```
...
Permission is hereby granted, free of charge, to any
person obtaining a copy of this software and associated
documentation files ( the "Software" ), to deal in the
Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to
do so, subject to the following conditions:

The above copyright notice and this permission notice shall
be included in all copies or substantial portions of the
Software.
...
```

Licenses can be basically grouped into two types - the permissive license and copyleft (protective) license. Some examples of the permissive license are MIT License, BSD licenses, and Apache license. A typical example of the copyleft ones is the GNU General Public License. When OSS reuses another OSS, if the reused OSS is licensed under a permissive license, the developed OSS does not need to open its source code. But if the reused OSS is licensed under a copyleft one, the developed OSS is enforced to open its source code. Usually, a permissive license violates a copyleft one.

For example, MIT license [*4], which is a permissive license, violates GPL-2.0+ license [*5], which is a copyleft one. GPL-2.0+ license declares:

```
...
This program is free software; you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation;
either version 2, or (at your option) any later version.
...
```

According to the terms of GPL-2.0+ license, if OSS licensed under MIT license reuses another OSS licensed under GPL-2.0+ license, illegal reuse occurs since OSS licensed under GPL-2.0+ license can only be redistributed and/or modified under the terms of the GNU General Public License either version 2, or (at your option) any later version as published by the Free Software Foundation.

Furthermore, some licenses share the same name but are with different versions. An example is the GNU General Public License. GNU General Public License has versions 1, 2, and 3. Each version has different terms and conditions. According to these different terms and conditions, three versions violate each other. Usually, licenses that share the same name are called a license family. For example, the GPL family includes GNU General Public License versions 1, 2, 3, and some other versions and MPL family includes Mozilla Public License versions 1, 1.1, 2.0, and some other versions.

2.2 Package Dependency

Introducing other packages in the user-contributed OSS ecosystem makes the package under development dependent on them. At the same time, the introduced packages may have their dependencies, seen as the indirect dependencies of the package under development. In this paper, a *direct dependency* of package A refers to package B whose name is explicitly recorded in package A's meta-data as its dependency. The *indirect dependencies*

[*2] https://www.npmjs.com
[*3] https://opensource.org/definition
[*4] https://opensource.org/licenses/MIT
[*5] https://opensource.org/licenses/GPL-2.0+

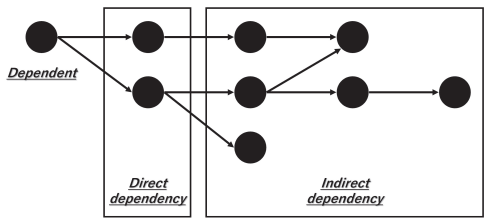


Fig. 1 The examples of direct dependency and indirect dependency.

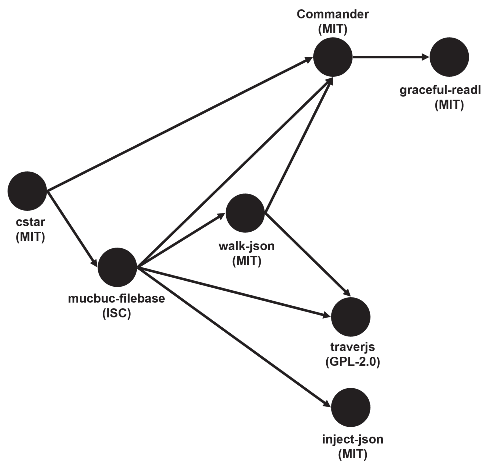


Fig. 2 A part of the dependency network of a `cstar` package in `npm`.

of package A refer to the transitive collection of package A's dependencies excluding package A's direct dependencies. **Figure 1** shows examples of direct dependency and indirect dependency.

Finally, direct dependencies and indirect dependencies could form a dependency network. The dependency network is prevalent in the user-contributed OSS ecosystem [4]. For example, the number of dependencies of packages in `npm` has grown 60% over 2016 and the mean number of the dependencies per package has grown to 60.1 [12]. **Figure 2** shows an example of the dependency network of a `cstar` package.

Developers are not likely to update the dependencies of their packages [2], [13]. Developers might also overlook the indirect dependencies since they do not include those dependencies by themselves [12]. These problems harm the health of the OSS ecosystem, causing problems such as security vulnerability, API breaking conflicts, and illegal reuse.

2.3 Dependency-related License Violation

License violation related to dependency occurs when a package's license violates any license of its dependencies. To achieve our research goal, we will define dependency-related license violation.

We first construct a license compatibility network to specify the compatibility between two licenses. Our license compatibility network is constructed based on the one created by David A. Wheeler [23]. Our license compatibility network is shown in **Figure 3**. Each arrow denotes one-directional compatibility. An arrow pointing from license A to license B means that license B does not violate license A. In other words, software licensed under license A is allowed to be combined into the software licensed under license B according to the proposed definition of violation in Section 1.

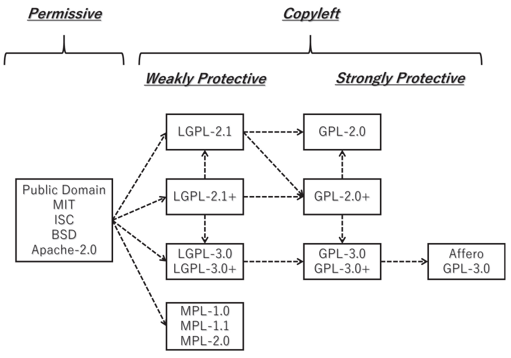


Fig. 3 License compatibility network.

In this paper, we define *dependency-related license violation* of a package X licensed under license A for another package Y licensed under license B, as the situation that Y is a dependency of X but no path from B to A exists in the license compatibility network (i.e., A violates B).

For example, `walk-json` package licensed under MIT license has dependency-related license violation for `traverjs` package licensed under GPL-2.0 license since `traverjs` package is a dependency of `walk-json` package, as shown in Fig. 2, and there is no path from GPL-2.0 license to MIT license in the license compatibility network (i.e., MIT license violates GPL-2.0 license), as shown in Fig. 3. This is an example of dependency-related license violation caused by direct dependency.

An example of dependency-related license violation caused by indirect dependency is `cstar` package. `cstar` package has two direct dependencies. The MIT license does not violate the licenses of these two direct dependencies. However, `traverjs` package licensed under GPL-2.0 license is also reused by `cstar` package as an indirect dependency. Because there is no path from GPL-2.0 license to MIT license in the license compatibility network, dependency-related license violation occurs in `cstar` package as well.

Note that our detection method proposed in Section 3.2 only detects dependency-related license violation using the dependency network and the license compatibility network constructed in this paper. Although we can define license violation in different ways, we use the above definition for simplicity and clarity of the implementation.

3. The Prevalence of Dependency-related License Violation

In this section, we aim to answer RQ1.

We first introduce the proposed method and then report the results of our empirical study.

3.1 Data Collection

We collect the target packages in `npm`. The observation period is from October 1st, 2010 to April 7th, 2017. We end up with 419,708 packages in total. Note that for each package, we only count it once no matter how many versions it has. We use the public API [*6] of `npm` to get the historical meta-data of all versions

[*6] https://registry.npmjs.org/-/all. Note that for some reason, `npm` has stopped providing this public API. Therefore, we cannot get the historical meta-data by using this API now.

of the target packages. For each version, the license and direct dependencies are recorded. For example, a part of the historical meta-data of `cstar` package is shown as follows:

```
...
        "version": "0.0.5",
        "dependencies":
        "commander": "*","mucbuc-filebase": "*",
        "license": "MIT",
...
        "version": "0.0.2",
        "dependencies":"mucbuc-filebase": "*",
        "license": "MIT",
...
```

3.2 Method

It is not easy to detect dependency-related license violations of a package in the user-contributed OSS ecosystem. There are three main challenges:

(1) A dependency is not always with the latest version, since packages in the user-contributed OSS ecosystem usually evolve frequently.
(2) Using dependencies is very common in the user-contributed OSS ecosystem. As a result, a package has a high probability of having a deep and complex dependency network.
(3) The same license is not always written in the same ways. For example, both GPL-2.0 and GPL version 2 refer to the same license.

To address these issues, we proposed a method that consists of five steps.

1) *Constructing the license dictionary:*

As mentioned above, the same license is usually written in different ways by different developers. We first construct a license dictionary, with which we can transform different forms of a license into a normalized one. The license dictionary includes the popular licenses published by the Open Source Initiative.

We collect all licenses written in the historical meta-data of the collected packages in `npm` and remove the duplicate one. We then use regular expression matching to do a preliminary classification. Regular expression matching is able to classify most licenses. We then manually check the results and move the license wrongly classified into the correct one. For the licenses which can not be matched by the regular expression, we manually classify them into the correct category. Note that since the license is written by the developer manually, some developers may not write the version number of the license. For example, some developers declare their packages are licensed under “GPL”, but no version is declared. For these cases, we record them as “license no version” such as “GPL no version” and “MPL no version”. “License no version” is seen as not violating any license in its family, but if a license violates any license in the family of “license no version”, it will be detected as violating “license no version”. For licenses that are not published by the Open Source Initiative, we classify them into a special category named “unknown” license.

All the normalized forms of our selected licenses are listed in **Table 1**. By constructing the license dictionary, we succeed in solving the third challenge - the various ways of writing the same license.

2) *Constructing the license compatibility network:*

We implement the license compatibility network proposed in Section 2.3. By searching this license compatibility network, we can check whether or not a license violates another one.

3) *Constructing the historical meta-data dataset:*

OSS in the user-contributed OSS ecosystem usually evolves frequently. With the evolution of software, the source code, license, and dependencies are also changing. However, OSS does not always reuse the latest version of its dependencies, thus deciding the proper version of dependencies is important. To accelerate the detection, we construct the historical meta-data dataset, in which the license and the direct dependencies with the proper versions of all historical versions of each package are recorded.

We have collected the historical meta-data of all versions of the target packages. However, it is necessary to process historical meta-data. Firstly, for licenses, we normalize them according to the license dictionary constructed in step 1. Secondly, for direct dependencies, we list all direct dependencies and choose the proper version for each direct dependency. Note that `npm` uses the semantic versioning standard to manage versions. The first release should start from 1.0.0. After this, changes should be handled according to **Table 2**. Developers can specify acceptable version ranges of the dependencies of their packages based on the semantic versioning standard used by `npm`. The specification is flexible by using different symbols to achieve different goals. **Table 3** shows some examples of how to specify the ranges. Note that we identify the version of a dependency as the latest version in the acceptable version ranges recorded in the meta-data of a package, which is consistent with the mechanism of `npm`. The details of the semantic versioning standard used by `npm` can be found in the documents of `npm`[*7].

After we process the historical meta-data, we can build the historical meta-data dataset for this package. The historical meta-data dataset includes information on license and direct dependencies of all versions of a package. **Table 4** shows the historical meta-data dataset constructed for `cstar` package in `npm`.

4) *Constructing the dependency network:* In this step, we construct the dependency network. The direct and indirect dependencies in the dependency network are attached to its version and license. Figure 2 shows a part of the dependency network constructed for `cstar` package in `npm`. For each dependency in the dependency network, the attached information of version and license is shown in **Table 5**. Note that the version of the `cstar` package is the latest version. The version of the direct dependency consists with the dependency’s version recorded in the meta-data of the `cstar` package. The version of indirect dependency is selected in a similar way.

By constructing the dependency networks for the packages in `npm`, we succeed in solving the first and the second challenges mentioned above - the variability of the dependencies’ versions and the complexity of the dependency networks.

5) *Dependency-related license violation detection:*

Since we have constructed the dependency network and the license compatibility network, it becomes possible to detect

[*7] https://docs.npmjs.com/getting-started/semantic-versioning

Table 1 The list of the selected licenses.

License	Normalized form	License family
Public Domain License	Public Domain	None
MIT License	MIT	None
ISC License	ISC	None
Apache License 2.0	Apache-2.0	None
3-clause BSD License/"New" or "Revised" license	BSD-3-Clause	BSD family
2-clause BSD License/"Simplified" or "FreeBSD" license	BSD-2-Clause	BSD family
Mozilla Public License version 1.0	MPL-1.0	MPL family
Mozilla Public License version 1.1	MPL-1.1	MPL family
Mozilla Public License version 2.0	MPL-2.0	MPL family
GNU General Public License version 2	GPL-2.0	GPL family
GNU General Public License version 2 or any later version	GPL-2.0+	GPL family
GNU General Public License version 3	GPL-3.0	GPL family
GNU General Public License version 3 or any later version	GPL-3.0+	GPL family
GNU Lesser General Public License version 2.1	LGPL-2.1	GPL family
GNU Lesser General Public License version 2.1 or any later version	LGPL-2.1+	GPL family
GNU Lesser General Public License version 3.0	LGPL-3.0	GPL family
GNU Lesser General Public License version 3.0 or any later version	LGPL-3.0+	GPL family
GNU Affero General Public License version 3	AGPL-3.0	GPL family

Table 2 The rules of how changes should be handled in `npm`.

CODE STATUS	STAGE	RULE	EXAMPLE
First Release	New Product	Start with 1.0.0	1.0.0
Bug fixes, other minor changes	Patch Release	Increment the third digit	1.0.1
New feature that don't break existing features	Minor release	Increment the middle digit	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit	2.0.0

Table 3 The examples of how to specify the ranges.

TYPE	EXAMPLE
All patch releases of major release 1.0	1.0 or 1.0.x or ˜1.0.0
All minor releases of major release 1	1 or 1.x or 1̂.0.0
All releases	* or x

Table 4 The historical meta-data dataset constructed for `cstar` package in `npm`.

Version	License	Direct dependency (Version)
0.0.5	MIT	commander (2.9.0), mucbuc-filebase (0.0.4)
0.0.4	MIT	commander (2.9.0), mucbuc-filebase (0.0.4)
0.0.3	MIT	mucbuc-filebase (0.0.4)
0.0.2	MIT	mucbuc-filebase (0.0.4)
0.0.1	MIT	mucbuc-filebase (0.0.4)
0.0.0	GPL-2.0	mucbuc-filebase (0.0.4)

Table 5 The attached information of version and license for the dependencies in the dependency network constructed for `cstar` package.

Dependency	Version	License
cstar	0.0.5	MIT
mucbuc-filebase	0.0.4	ISC
walk-json	0.0.2	MIT
commander	2.9.0	MIT
graceful-readlink	1.0.1	MIT
traverjs	0.0.7	GPL-2.0
inject-json	0.0.9	MIT

dependency-related license violations. We detect whether or not the license of this package violate the licenses of direct and indirect dependencies in the dependency network according to the license compatibility network. Dependency-related license violation is detected when the violation is found.

For example, by observing the dependency network constructed for `cstar` package (Fig. 2 and Table 5), we can find that MIT license - the license of `cstar` package - violates GPL-2.0 license - the license of `traverjs` package which is an indirect dependency. Therefore, `cstar` package is detected as having a dependency-related license violation.

The details of the proposed method can be found here [16].

Table 6 The top 10 dependency-related license violations.

Package	Dependence	Number	Proportion
MIT	GPL-3.0	582	16.1%
MIT	LGPL-3.0	349	9.6%
Public Domain	GPL-3.0	300	8.3%
MIT	LGPL no version	281	7.8%
ISC	GPL-3.0	231	6.4%
MIT	GPL no version	224	6.2%
MIT	LGPL-2.1	159	4.4%
MIT	GPL-2.0	134	3.7%
Public Domain	GPL no version	114	3.1%
Public Domain	LGPL-3.0	104	2.9%

3.3 Results and Discussion

We detect dependency-related license violations in the collected 419,708 packages in `npm`. As a result, only 2,704 packages are detected as having dependency-related license violations, accounting for 0.644% of all packages. The result suggests that only a few packages (0.644%) in `npm have dependency-related license violations.` We also observe that among these 2,704 packages, 3,624 dependency-related license violations are detected. **Table 6** shows the top 10 list of those violations classified by licenses. The result shows that most dependency-related license violations are caused by the violation between the permissive licenses and copyleft licenses, while usually copyleft licenses do not violate each other.

A potential reason is the developers' manner of choosing licenses for their packages. To ascertain this reason, we count the proportion of the different licenses in `npm`. **Table 7** shows the result. We observe that the permissive licenses take a large part of all licenses while the copyleft licenses are not widely used in `npm`. The preference of the permissive licenses may be the reason for the low proportion of the packages detected as having dependency-related license violations in `npm`.

Furthermore, because of the high proportion of the permissive licenses and low proportion of the copyleft licenses used in

Table 7 The proportion of the selected licenses in `npm`.

License	Number	Proportion
MIT	254,972	60.75%
None	80,331	19.14%
ISC	33,827	8.06%
Apache-2.0	13,598	3.24%
BSD family	17,794	4.24%
GPL family	9,158	2.18%
Unlicense/Public Domain	2,098	0.50%
MPL family	1,132	0.27%
Unknown	6,798	1.62%
All	419,708	100%

`npm`, we could assume that including the packages licensed under copyleft licenses in the dependency network potentially causes a high dependency-related license violation. To ascertain it, we selected GPL family licenses as the target to study. We collected the packages having direct or indirect dependencies with GPL family licenses. As a result, we collected 4,067 packages. Among them, 2,704 packages are detected as having dependency-related license violations, accounting for 66.84%. Note that all packages detected as having dependency-related license violations in the detection of 419,708 packages are included in these 4,067 packages. The result proves our assumption. A possible explanation is that the developers do not understand the copyright notice well and overlook the license violation when they use a package as a dependency. Another possible explanation is that the developers overlook the indirect dependencies since they do not include those packages by themselves. To ascertain it, we calculate the proportion of the dependency-related license violations caused by direct and indirect dependency respectively. As a result, among 2,704 packages detected as having dependency-related license violations, 2,115 packages are detected as having violations caused by direct dependencies, accounting for 78.2%. Meanwhile, 1,507 packages are detected as having violations caused by indirect dependencies, accounting for 55.7%. The result suggests that both direct dependency and indirect dependency play an important role in the occurrence of dependency-related license violations.

We will study more on developers' attitudes in Section 4.

Hence, we answer RQ1:

> Only a few packages (0.644%) in npm have dependency-related license violations. However, including the packages licensed under copyleft licenses as dependency is closely related with the occurrence of dependency-related license violations.

4. Preliminary Questionnaire

In this section, we aim to answer RQ2.

We first introduce the preliminary questionnaire we conducted and then report the results.

4.1 Questionnaire Design

Our preliminary questionnaire targets the authors of packages detected as having dependency-related license violations. Therefore, our questionnaire first describes how the license of the target package violates the licenses of its dependencies by explaining the terms and conditions of the licenses. The second part of the questionnaire then asks developer opinions on the following two questions: 1) *Do you think this is a kind of risk? If so, were you aware of this kind of risk when you were developing your packages?* 2) *In this question, you can share anything you want to say with this kind of risk with us.*

For the analysis, we first record the responses of the first question according to whether or not the developer thinks the dependency-related license violations in their packages to be issues. We then analyze the responses of the second question through a systematic method: (i) reading of each response, (ii) checking, summarizing and categorizing text, and (iii) looking for similarities or differences in other responses. We finally end up with a list of important observations in the responses.

4.2 Data Collection

In Section 3, 2,704 packages are detected as having dependency-related license violations. We randomly select 100 packages and send their authors email invitations for our preliminary questionnaire. In the end, we received 20 responses which equals a response rate of 20%.

4.3 Results and Discussion

4.3.1 Question 1

For the first question in our questionnaire, 11 participants out of 20 participants regard the dependency-related license violations as risks. On the contrary, 6 participants do not regard the dependency-related license violations as risks and 3 participants are not sure whether or not the dependency-related license violations are risks respectively. The results suggest the divergence of developers towards dependency-related license violations. To study the reasons why developers regard or do not regard the dependency-related license violations as risks, we also analyze the developers' explanations for their choices.

Among 11 participants regarding the dependency-related license violations as risks out of all 20 participants, 5 participants were aware of the dependency-related license violation issue when they were developing their packages while 6 participants were not. Only one participant out of 6 participants who was not aware of this issue explains the reason: as an individual developer, this participant will not take time to properly study the licenses in dependencies, especially when the package is not developed for commercial use. Meanwhile, 4 participants out of 5 participants who were aware of this issue explain the reasons: 1) Developers will not check the licenses of the dependencies for personal projects. 2) The project is weakly maintained so the license violation issue is not addressed carefully. 3) The developer is the author of both the detected package and its dependency. 4) The developer knows the license violation issue but is not well informed. The results suggest that the developers potentially overlook the dependency-related license violations even though they regard them as risks.

All 6 participants who do not regard the dependency-related license violations as risks out of all 20 participants explain their reasons. There are two main reasons: 1) There is no need to care about the license violation issue if the package is not developed for commercial use. 2) Reusing other packages as dependencies is not "redistribution".

The first reason is obviously wrong since the terms and conditions in licenses are effective whether the package is developed for commercial use or not. For the second reason, judging whether or not reusing other packages as dependencies is "redistribution" is a complicated legal problem and may differ in different countries. However, even if reusing other packages as dependencies is not regarded as "redistribution", it could potentially become a risk for the end-users who will reuse these packages for various purposes.

The results reveal the developers' overlooking and misunderstanding of the dependency-related license violations.

4.3.2 Question 2

We analyze the responses of the second question through a systematic method and make the following important observations:

1) *It is not easy to understand the terms and conditions of the licenses.* Most participants describe their understanding of the terms and conditions of the licenses. For the same license, participants may have a different understanding. The most important problem is understanding the meaning of particular words in the licenses such as "redistribute", "reuse", and "modify". It is also difficult for developers to identify the difference between two licenses in the same license family, such as "GPL-2.1" and "GPL-3.0". The difficulties in understanding the terms and conditions also hinder developers in choosing the proper licenses.

2) *Managing dependency-related license violations is difficult in practice.* Some participants describe the difficulty in managing dependency-related license violations in practice. One participant says that a lot of open source components are reused in his project, and the collection of dependencies grows and changes during the life cycle of the project. He usually generates reports on the licensing requirements for dependencies manually and thus it takes a lot of time. Another participant says that he always has to re-write his packages because of the license of the dependencies. Note that the licenses of the dependencies may change during the life cycle and it is difficult to trace them [13]. The dependency-related license violations may also occur when a developer is required to change the license of his package to meet the requirement of the end-users.

3) *Help in managing the dependency-related license violations is demanded.* Some participants agree that tools that help to manage the dependency-related license violations will help a lot. The following functions are wanted: 1) detecting the dependency-related license violations; 2) tracing the change of the licenses of the dependencies; 3) choosing the proper license.

The results highlight the difficulties in managing dependency-related license violations and the developers' demands for help.

Hence, we answer RQ2:

> The attitudes of developers towards dependency-related license violations vary. The dependency-related license violations are overlooked and misunderstood by the developers for various reasons. Managing dependency-related license violations is difficult and the developers are demanding help.

5. Threats to Validity

This section discusses the threats to the validity of our research. Threats to construct validity concern the relationship between theory and outcome, and relate to possible measurement imprecision when extracting data we used in this study. `npm` hosts over 1.3 million JavaScript packages. This number is still increasing rapidly every day. It is impossible to conduct the empirical study on packages of this large number, especially considering that the packages depend on each other. Otherwise, because of the rapid increment of the packages in `npm`, the public API we use to get the historical meta-data is not accessible. The collected data is the only one we can use to study dependency-related license violations in `npm`. We set the observation period from October 1st, 2010 to April 7th, 2017, and collected the meta-data of 419,708 packages in total. 419,708 packages are enough to represent all packages in `npm` statistically. Furthermore, these 419,708 packages only depend on each other and do not depend on other packages in `npm`, which makes the empirical study executable.

Threats to internal validity concern factors internal to the study that could impact our results. Such threat does not affect an exploratory study like the one in this paper. The only case worthwhile being discussed is our answer to RQ2, where we observe and understand the responses manually based on our knowledge.

Threats to external validity are related to the ability to generalize the findings in our study. Our empirical study is only conducted on `npm`. `npm` is one of the most popular and successful OSS ecosystems, which makes itself a suitable target to study copyright inconsistencies. However, we find that the developers in `npm` prefer not to choose copyleft licenses for their packages, which is a possible reason for the low proportion of packages having dependency-related license violations. But according to the different situations, other OSS ecosystems may have different results. As the first work focusing on dependency-related license violations in OSS ecosystems, our findings could be a good baseline to study dependency-related license violations in them. We agree that it is necessary to replicate our empirical study on different OSS ecosystems.

Another threat worth discussion is the scale of our preliminary questionnaire. Our preliminary questionnaire only selects 100 packages from 2,704 packages detected as having dependency-related license violations. Compared with the large numbers of packages and developers in `npm`, the scales of the target packages and developers are not enough to achieve a result with statistical sense. However, as a preliminary qualitative analysis of dependency-related license violations, it can still reveal developers' attitudes, some of which are important. But we also agree that a large-scale developer survey is necessary in our future works.

6. Related Work

6.1 Software License

Many studies in software engineering investigate software licenses. There are some studies that are devoted to identifying licenses [7], [10], [18]. Based on these studies, some researchers analyzed software licenses in open source projects and revealed some license issues. Di Penta et al. [5] provided an automatic method to track changes occurring in the licensing terms of a system and did an exploratory study on license evolution in six open source systems and explained the impact of such evolution on the projects. German et al. [8] proposed a method for understanding licensing compatibility issues in software packages. They mainly focused on the compatibility between licenses declared in packages and those in source files. Different from their work, we mainly focused on the compatibility between licenses declared in packages and its dependencies in this paper. In another research by Di Penta et al. [9], they analyzed license inconsistencies of code siblings (a code clone that evolves in a different system than the code from which it originates) between Linux, FreeBSD, and OpenBSD, but they did not explain the reasons underlying these inconsistencies. Wu et al. proposed an approach to find license inconsistencies in similar files [24]. By investigating the revision history of these files, they summarized the factors that caused these license inconsistencies and tried to decide whether they are legally safe or not. References [9] and [24] focused on the license violations on the source file level and source code level, while we focused on the package level. Alspaugh et al. [1] proposed an approach for calculating conflicts between licenses in terms of their conditions. Vendome et al. [21] performed a large empirical study of Java applications and found that changing a license is a common event and that there is a lack of traceability between when and why the license of a system changes. Vendome et al. performed a study on GitHub and found that developers adopting a license may depend on various factors and they discovered the lack of traceability of when and why licensing changes are made and highlighted the need for better tools to support in guiding developers in choosing and changing licenses and in keeping track of the rationale of license changes [20].

6.2 License Compliance

License compliance is an important area of research that draws attention from many researchers. Zhang et al. have developed a tool named LCheck that utilizes Google Code Search service to check whether a local file exists in an OSS project and whether the licenses are compatible [25]. German et al. proposed a tool named Kenen that checks license compliance for Java components that uses component identification, provenance discovery, license identification, and licensing requirements analysis [6]. Van der Burg et al. proposed an approach that can uncover license compliance inconsistencies by analyzing the Concrete Build Dependency Graph of a software system [19]. They proposed an approach to construct and analyze the Concrete Build Dependency Graph of a software system by tracing system calls that occur at build-time. Kapitsaki et al. proposed an approach of automating license compliance with a process that examines the structure of Software Packages Data Exchange [11]. Different from the above works, we mainly focused on dependency-related license violations in OSS ecosystems. We utilize the meta-data of the packages to detect license violations, which is an important characteristic of OSS ecosystems. Vendome et al. studied the rationale of developers in choosing and changing licenses and investigated the problem of traceability of license changes [22]. They provided a vision of ensuring license compliance of a system.

7. Conclusion

In this paper, we propose a method to detect dependency-related license violations in OSS ecosystems, with which we conduct an empirical study on `npm` to study the prevalence of dependency-related license violations. The result suggests that only a few packages (0.644%) in `npm have dependency-related license violations,` but including the packages licensed under copyleft licenses in the dependency network still potentially causes a high dependency-related license violation. We also conduct a preliminary questionnaire on the authors of packages detected as having dependency-related license violations, revealing the developers' overlooking and misunderstanding of the dependency-related license violations, the difficulties in managing dependency-related license violations, and the developers' demands for help. Our work highlights the importance of the dependency-related license violation issue, and also creates the possibility of studying dependency-related license violations in OSS ecosystems, which is overlooked by the software engineering researchers.

In our future work, we plan to extend our study on dependency-related license violations to other OSS ecosystems. Furthermore, based on the results of the preliminary questionnaire, we also consider a new large-scale developer survey as our future work. We also plan to implement some tools to help developers in maintaining licenses and managing license violations.

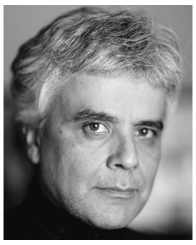
Acknowledgments This work was supported by JSPS KAKENHI Grant Number 18H04094.

References

[1] Alspaugh, T., Asuncion, H. and Scacchi, W.: Intellectual Property Rights Requirements for Heterogeneously-Licensed Systems, *Proc. 17th International Requirements Engineering Conference* (*RE2009*), pp.24–33 (2009).
[2] Bavota, G., Canfora, G., Di Penta, M., Oliveto, R. and Panichella, S.: How the Apache community upgrades dependencies: An evolutionary study, *Empirical Software Engineering*, Vol.20, No.5, pp.1275–1317 (2015).
[3] Boehm, B.W.: Improving Software Productivity, *Computer*, Vol.20, No.9, pp.43–57 (1987).
[4] Decan, A., Mens, T. and Grosjean, P.: An empirical comparison of dependency network evolution in seven software packaging ecosystems, *Empirical Software Engineering*, Vol.24, No.1, pp.381–416 (2019).
[5] Di Penta, M., German, D.M., Guéhéneuc, Y.-G. and Antoniol, G.: An Exploratory Study of the Evolution of Software Licensing, *Proc. 32nd International Conference on Software Engineering* (*ICSE2010*), pp.145–154 (2010).
[6] German, D. and Di Penta, M.: A method for open source license compliance of java applications, *IEEE Software*, Vol.29, No.3, pp.58–63 (2012).
[7] German, D.M., Manabe, Y. and Inoue, K.: A sentence-matching method for automatic license identification of source code files, *Proc. 25th International Conference on Automated Software Engineering* (*ASE2010*), pp.437–446 (2010).
[8] German, D., Di Penta, M. and Davies, J.: Understanding and Auditing the Licensing of Open Source Software Distributions, *Proc. 18th International Conference on Program Comprehension* (*ICPC2010*), pp.84–93 (2010).
[9] German, D., Di Penta, M., Gueheneuc, Y.-G. and Antoniol, G.: Code siblings: Technical and legal implications of copying code between applications, *Proc. 6th Working Conference on Mining Software Repositories* (*MSR2009*), pp.81–90 (2009).
[10] Gobeille, R.: The FOSSology Project, *Proc. 5th Working Conference on Mining Software Repositories* (*MSR2008*), pp.47–50 (2008).
[11] Kapitsaki, G.M., Kramer, F. and Tselikas, N.D.: Automating the license compatibility process in open source software with SPDX, *Jour-*

nal of Systems and Software, Vol.131, pp.386–401 (2017).
[12] Kikas, R., Gousios, G., Dumas, M. and Pfahl, D.: Structure and Evolution of Package Dependency Networks, *Proc. IEEE/ACM 14th International Conference on Mining Software Repositories* (*MSR2017*), pp.102–112 (2017).
[13] Kula, R.G., German, D.M., Ouni, A., Ishio, T. and Inoue, K.: Do developers update their library dependencies?, *Empirical Software Engineering*, Vol.23, No.1, pp.384–417 (2018).
[14] Lungu, M., Lanza, M., Gîrba, T. and Robbes, R.: The small project observatory: Visualizing software ecosystems, *Science of Computer Programming*, Vol.75, No.4, pp.264–275 (2010).
[15] McIlroy, M.D., Buxton, J., Naur, P. and Randell, B.: Mass-produced software components, *Proc. 1st International Conference on Software Engineering* (*ICSE1968*), pp.88–98 (1968).
[16] Qiu, S.: Empirical Studies on License Compliance and Copyright Inconsistency Risks in Open Source Software, Master's thesis, Osaka University, Japan (2018).
[17] Standish, T.A.: An Essay on Software Reuse, *IEEE Trans. Software Engineering*, Vol.SE-10, No.5, pp.494–497 (1984).
[18] Tuunanen, T., Koskinen, J. and Kärkkäinen, T.: Automated software license analysis, *Automated Software Engineering*, Vol.16, No.3-4, pp.455–490 (2009).
[19] Van der Burg, S., Dolstra, E., McIntosh, S., Davies, J., German, D.M. and Hemel, A.: Tracing software build processes to uncover license compliance inconsistencies, *Proc. 29th ACM/IEEE International Conference on Automated Software Engineering* (*ASE2014*), pp.731–742, ACM (2014).
[20] Vendome, C., Bavota, G., Di Penta, M., Linares-Vásquez, M., German, D. and Poshyvanyk, D.: License usage and changes: A large-scale study on gitHub, *Empirical Software Engineering*, Vol.22, No.3, pp.1537–1577 (2017).
[21] Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., Germán, D.M. and Poshyvanyk, D.: License Usage and Changes: A Large-Scale Study of Java Projects on GitHub, *Proc. 23rd IEEE International Conference on Program Comprehension* (*ICPC2015*) (2015).
[22] Vendome, C. and Poshyvanyk, D.: Assisting developers with license compliance, *Proc. 38th International Conference on Software Engineering Companion*, pp.811–814, ACM (2016).
[23] Wheeler, D.A.: The Free-Libre/Open Source Software (FLOSS) License Slide (2017), available from ⟨https://www.dwheeler.com/essays/floss-license-slide.html⟩.
[24] Wu, Y., Manabe, Y., Kanda, T., German, D.M. and Inoue, K.: A method to detect license inconsistencies in large-scale open source projects, *Proc. 12th Working Conference on Mining Software Repositories* (*MSR2015*), pp.324–333 (2015).
[25] Zhang, H., Shi, B. and Zhang, L.: Automatic checking of license compliance, *Proc. 2010 IEEE International Conference on Software Maintenance* (*ICSM2010*), pp.1–3, IEEE (2010).

Shi Qiu received his B.E. degree of software engineering from Jilin University in 2013 and his M.E. degree of information science and technology from Osaka University in 2017. At present, he is a Ph.D. student in the Graduate School of Information Science and Technology at Osaka University. His research interests include mining software repositories, software license/copyright analysis, and software ecosystem.



Daniel M. German is Professor in the Department of Computer Science at the University of Victoria, where he does research in the areas of mining software repositories, open source software ecosystems and the impact of intellectual property in software engineering.

Katsuro Inoue received his Ph.D. from Osaka University in 1984. He was an associate professor of the University of Hawaii at Manoa from 1984 to 1986. After becoming an assistant professor of Osaka University in 1986, he has been a professor since 1995. His research interests include software engineering, especially software maintenance, software reuse, empirical approach, program analysis, code clone detection, and software license/copyright analysis.