修士学位論文

題目

Empirical Studies on License Compliance and Copyright Inconsistency Risks in Open Source Software

指導教員

井上 克郎 教授

報告者

仇　実

平成 30 年 2 月 7 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻 ソフトウェア工学講座

平成 29 年度 修士学位論文

Empirical Studies on License Compliance and Copyright Inconsistency Risks in Open Source Software

仇　実

内容梗概

Open source software (OSS) is computer 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. In addition, the name of authors, also called copyright holders, are usually included in the header part of source file as well.

A license may be incompatible with another one according to the terms and conditions. Making software by reusing OSS software may cause license compliance risks if the developers did not notice the license of reused OSS software. Meanwhile, the inconsistency of main contributor and copyright holder may also potentially cause legal risks.

In this paper, we propose a method to detect license compliance risk and conduct an empirical study on NPM, a JavaScript-based software ecosystem. For copyright inconsistency risks, we select Linux kernel as our target to find the files of which the main contributors are inconsistent with copyright holders. We also conduct an evolutionary study to understand the change of copyright inconsistency risks in Linux kernel. In this study, we propose a method to detect license compliance risks and copyright inconsistency risks. Our result shows that although the proportion of packages detected as having license compliance risk is quite low, the reuse of packages licensed under the copyleft license is still more likely to cause license compliance risk. We also reveal the prevalence of copyright inconsistency risks.

主な用語

Open source software
Software license
Software copyright

目 次

1 Introduction

Software reuse has long been proved to be a good method to increase software productivity [11, 12, 2]. As a practice, reusing open source software (OSS) has become more and more popular. The reused OSS software is usually called the dependencies of the software under development. When developers reusing OSS software, they should pay special attention to open source license and software copyright to prevent legal risk [19].

1.1 Open source license

Open source license describes the terms and conditions when OSS software is used, modified and shared. OSS software 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].

Many studies in software engineering have been done on software license. Some effective approaches and tools are proposed to identify the license of source code files automatically [6].

Just as the definition of open source license says, OSS software can only be reused as long as the particular terms and conditions are satisfied. Therefore, the developed software should satisfy all terms and conditions in licenses of all its dependencies. In other words, the developed software should select a license which is compatible with the licenses of all dependencies. If the selected license is incompatible with any license of its dependencies, potential legal risks may occur. Here we define this potential legal risk as license compliance risk.

With the rise of user-contributed OSS ecosystems, judging whether the terms and conditions are satisfied or not has become a serious issue. OSS ecosystem consists of software projects that are developed and evolve together in a shared environment [10]. User-contributed OSS ecosystem is an ecosystem where software projects are contributed by its users.

In a user-contributed OSS ecosystem, it's much more difficult to judge whether the selected license is compatible with any license of its dependencies or not. To address this issue, we proposed an approach to detect license compliance risks of software projects in

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

such OSS ecosystems. We are also interested in the current situation of user-contributed OSS ecosystems.

We select `npm`[2] as the target in our research. `npm` serves as a large repository of JavaScript-based software packages. It hosts over 450,000 JavaScript packages and is the largest JavaScript ecosystem, with millions of packages being installed from the `npm` repository on an everyday basis. We also conduct an empirical study on `npm` with our proposed method to understand the prevalence of license compliance risk. We discovered that the packages detected as having direct or indirect dependency risk take only a small portion of all the packages in `npm`. We reveal that reuse of packages licensed under the copyleft license is more likely to cause license compliance risk. We also discovered some characteristics of license compliance risk in `npm`.

The contributions of this study are:

1. We proposed an approach to detect the license compliance risk of software projects in OSS ecosystems, which can expose all licenses in dependencies incompatible with the license of detected software projects.

2. We applied our method to `npm` and conducted an empirical study to understand the current situation of the license compliance risk in user-contributed OSS ecosystem.

1.2 Copyright holder

Software copyright is a special case of copyright, which is used to prevents the unauthorized copying of software. OSS software is also protected by software copyright. Software copyright is written in the header part of source code as well. However, since the OSS software is usually contributed by more than one contributors, it's very difficult to judge who are the real copyright holders. In other words, there is no accurate definition of copyright holders. Could any contributor be a copyright holder even they contributed only a few lines of code? Could a developer claim himself as a copyright holder even if his copyright information is not included in the header part of the source code?

Actually, there are some real-world legal cases related to copyright. Recent years, some contributors of Linux kernel[3] complained about some companies. Those companies have to compensate the contributors because they didn't obey the terms and conditions of copyright. Here comes a problem, how companies judge whether these developers are

[2]`https://www.npmjs.com`
[3]`https://github.com/torvalds/linux`

the real copyright holders as what they have declared? On the one hand, the copyright holders whose information are written in the header part of source code are usually treated as the real copyright holders; on the other hand, some main contributors who contributed the large part of source code should also be regarded as copyright holders according to the common sense even their information are not written in the header part of source code. Therefore, the inconsistency between the declared copyright holders and the main contributors has become a risk. It makes companies or developers not able to recognize the real copyright holders when they reuse the OSS software.

In this study, we propose an approach to detect the copyright inconsistency risk, known as the inconsistency between the declared copyright holders and the main contributors. Then we applied our approach in Linux kernel and conduct an empirical study to understand the prevalence of copyright inconsistency risk in OSS projects. The result shows that the proportion of source code files having the copyright inconsistency risk is quite high. Then with an analysis of some source code files having the copyright inconsistency risk, we discovered that there are some reasons behind the copyright inconsistency risk in Linux kernel.

The contributions of this study are:

1. We proposed an approach to detect the copyright inconsistency risk of software projects in OSS ecosystem.

2. We applied our method to Linux kernel and conducted an empirical study to understand the current situation of the copyright inconsistency risk in user-contributed OSS ecosystem.

1.3 Structure of this paper

This paper is organized as follows. Section 2 describes background on license compliance and definite license compliance risk. Section 2 also shows some examples of packages with license compliance risk. Section 3 introduces our method of detecting license compliance risk. An empirical study with this method is described in Section 4. Section 5 describes the background of copyright and the definition of copyright inconsistency risk. Section 6 introduces our method of detecting copyright inconsistency risk, followed by section 7 with an empirical study with this method and the discussion of the results. Related work is described in Section 8, after which section 9 concludes this paper and points out the future work.

2 License compliance risk

2.1 License compliance

A software license permits a software to be reused under some terms and conditions. An open source license is a software license that follows Open Source Definition[4] and is approved by Open Source Initiative. A software license is written in the header part of source code. Here is an example of a license statement taken from grunt[5] package in `npm`, which states that the file is under the 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.
[...]
```

Open Source Initiative does not only determine the definition of open source license but also publish the list that includes all approved licenses are available. Open Source Initiative also declares that the following OSI-approved licenses are popular and widely used among all licenses[6]:

```
Apache License 2.0
BSD 3-Clause "New" or "Revised" license
BSD 2-Clause "Simplified" or "FreeBSD" license
GNU General Public License (GPL)
GNU Library or "Lesser" General Public License (LGPL)
MIT license
Mozilla Public License 2.0
Common Development and Distribution License
Eclipse Public License
```

To definite license compliance risk, we should first show how a license is not compatible with another one. Licenses can be basically grouped into two types - permissive license and

[4]https://opensource.org/definition
[5]https://www.npmjs.com/package/grunt
[6]https://opensource.org/licenses

copyleft license. Some examples of the permissive license are MIT License, BSD licenses, Apple Public Source License and Apache license. While GNU General Public License is the typical examples of the copyleft ones. When a developed OSS software reuses another OSS software, if the reused OSS software is licensed under a permissive license, the developed OSS software does not need to open its source code. While if the reused OSS software is licensed under a copyleft one, the developed OSS software is enforced to open its source code. Usually, a permissive license is not compatible with a copyleft one. Here is an example: The following texture is a part of Apache-2.0 license:

```
[...]
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
[...]
```

While GPL-2.0+ license says:

```
[...]
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 an OSS software licensed under Apache-2.0 license reuse an OSS software licensed under GPL-2.0+ license, an illegal reuse occurs since OSS software licensed under GPL-2.0+ license can only be redistributed and/or modified 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.

Note that some licenses share the same name but with different versions. An example is 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 are not compatible with each other. Here is an example:

GPL-3.0+ license says:

```
[...]
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 3 of the License, or (at your option) any
later version.
[...]
```

Although GPL-2.0+ license and GPL-3.0+ license are grouped under the same name, it's obvious that they are not compatible with each other. After understanding how a license is not compatible with another one, we introduce our definition of license compliance risk and give some examples of license compliance risks we have found in `npm`.

2.2 Definition

License compliance risk refers to the situation that the license of an OSS software is not compatible with the license of its dependency.

2.3 Example

By analyzing `npm`, we observed some cases of license compliance risk.

The first one is `nodeos-bootfs`[7] package, which reuses another package called `genfatfs`[8]. `nodeos-bootfs` package declares its license as MIT license in meta file:

```
[...]
        "author": "Jesus Leganes Combarro 'piranna' <piranna@gmail.com>",
        "license": "MIT",
        "dependencies": {
                "nodeos-cross-toolchain": "^1.0.0-RC3.0",
                "download-manager": "^0.1.3",
                "genfatfs": "^1.0.3"
        },
[...]
```

While `genfatfs` package declares its license as GPL-2.0 license in the same way:

```
[...]
        "contributors": [
                "Jesus Leganes Combarro 'piranna' <piranna@gmail.com>"
        ],
        "license": "GPL-2.0",
```

[7]https://www.npmjs.com/package/nodeos-bootfs

[8]https://www.npmjs.com/package/genfatfs

```
cstar (MIT)
        |= commander (MIT)
        |       |= graceful-readlink (MIT)
        |= mucbuc-filebase (ISC)
        |       |= walk-json (MIT)
        |       |       |= traverjs (GPL-2.0)
        |       |= inject-json (MIT)
```

Figure 1: The dependency chain of `cstar` package.

```
        "bugs": {
                "url": "https://github.com/NodeOS/genfatfs/issues"
        },
[...]
```

Because MIT license is not compatible with GPL-2.0 license, `nodeos-bootfs` package is reported as having license compliance risk.

Another example is `cstar`[9] package. Different from `nodeos-bootfs` package, `cstar` package declares its license as MIT license:

```
[...]
        "author": "mark busenitz",
        "license": "MIT",
        "bugs": {
                "url": "https://github.com/mucbuc/cstar/issues"
        },
[...]
```

This license is compatible with licenses of all `cstar` package's dependencies. But by observing the dependency chain of `cstar` package in Figure 1, another package called `traverjs`[10] is also reused by `cstar` package. `traverjs` package is also licensed under GPL-2.0 license:

```
[...]
        "author": "mbusenitz",
        "license": "GPL-2.0",
        "bugs": {
                "url": "https://github.com/mucbuc/traverjs/issues"
        },
[...]
```

[9] https://www.npmjs.com/package/cstar

[10] https://www.npmjs.com/package/traverjs

It is obvious that MIT license is not compatible with GPL-2.0 license, so `cstar` package is also reported as having license compliance risk.

The examples above are all license compliance risks at the package level. However, license compliance risk can also occur at file level. Here is an example. `hulq`[11] package is licensed under Apache-2.0 license:

```
[...]
        "author": "Lucas Chain",
        "license": "Apache-2.0",
        "dependencies": {
                "chalk": "^1.1.0",
                "gulp": "^3.9.0",
[...]
```

The source code files of `hulq` package include a file called tap-driver, of which the license is written in the header part of source code:

```
[...]
        # Copyright (C) 2011-2013 Free Software Foundation, Inc.
        #
        # 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.
[...]
```

tap-driver file is licensed under GPL-2.0+ license. Since Apache-2.0 license is not compatible with GPL-2.0+ license, this case is also regarded as license compliance risk.

2.4 Categorization

Based on the analysis of above examples, we categorize all license compliance risks into three types:

1. **Direct dependency risk**:

   The license of an OSS software is not compatible with the license of its direct dependency.

2. **Indirect dependency risk**:

   The license of an OSS software is not compatible with the license of one OSS software in its dependency chain.

[11] https://www.npmjs.com/package/hulq

3. **Self risk**:

   The license of an OSS software is not compatible with one source code file of OSS software itself.

3 Detection of license compliance risk

3.1 Problems

It is difficult to detect license compliance risk of an OSS software in user-contributed Open Source Software (OSS) ecosystem. The following aspects are some main challenges:

1. OSS software in user-contributed OSS ecosystem usually evolves frequently, because of which one OSS software usually reuse another one of old version instead of the latest one.

2. Software reuse is very popular in user-contributed OSS ecosystem. As a result, an OSS software has a high probability of having a deep and complex dependency chain.

3. The license of an OSS software in user-contributed OSS ecosystem is usually written in a license file or recorded in the meta file. But the same license is usually written in different terms. For example, both GPL-2.0 and GPL version 2 refer to the same license.

4. These challenges make the detection of license compliance risk a difficult and complex problem.

3.2 Data collection

We select `npm` as our target ecosystem to detect license compliance risk. `npm` serves as a large repository of JavaScript-based software packages. It hosts over 450,000 JavaScript packages to become the largest software ecosystem, with millions of packages being installed from the `npm` repository on an everyday basis. As shown in Figure 2, when a package is installed from npm, those packages which are in the dependency chain of this package are also installed and included into the source code of the project. This is the mechanism of software reuse in `npm`.

In the first step, we need to collect empirical data that represents popularity and other main software ecosystem factors for `npm`. Table 1 shows a summary of collected npm packages. The observation period is from October 1st, 2010 to April 7th, 2017, and all data we collected only cover this range as well. At last, we collect 419,708 packages. We end up with 419,708 packages in total.

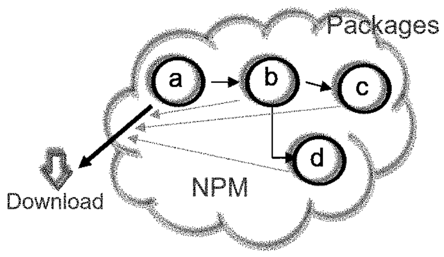


Figure 2: The effect of dependencies on downloads in `npm`. When package A is installed, package B and C are downloaded at the same time.

Table 1: Summary Statistics of the collected dataset.

	Dataset statistics
observation period	2010-Oct to 2017-Apr
packages	419,708

3.3 Method

To address the challenges mentioned, our method is divided into 5 steps:

1. **Build the license dictionary**

   As we mentioned, the same license is usually written in different terms by different developers. Take GPL-2.0 license for an example. The common style is GPL-2.0, but we find a lot of styles of writing by observing the licenses written in the meta file of packages in `npm`, such as GPL-2, GPLv2, GPL 2, GNU GPL-2.0, GPL version 2, etc. All these terms refer to GPL-2.0 license. To detect license compliance risk, all these terms should be classified into a normal one. We name this classification as table license dictionary.

   To build the license dictionary, we first select 19 popular licenses as keys. Note that the following detection is also only conducted on these licenses. All our selected licenses are listed as follows:

```
Public Domain
MIT and X11 License
ISC License
Apache-2.0 (Apache License 2.0)
BSD-3-Clause (2-clause BSD License)
BSD-2-Clause (3-clause BSD License)
MPL (Mozilla Public License) family (MPL-1.0, MPL-1.1, MPL-2.0)
```

```
GPL-2.0 (GNU General Public License version 2) and GPL-2.0+
GPL-3.0 (GNU General Public License version 3) and GPL-3.0+
LGPL-2.1 (GNU Lesser General Public License version 2.1) and LGPL-2.1+
LGPL-3.0 (GNU Lesser General Public License version 3.0) and LGPL-3.0+
AGPL-3.0 (GNU Affero General Public License version 3)
```

We collect all licenses written in the meta file of packages in `npm` and drop the duplicate one. Then we use regular expression matching to do a preliminary classification. Regular expression matching success in classifying quite a part licenses. Then we manually check the results and move the license which is wrongly classified into the right one. For the licenses which can't be matched by regular expression, we manually classified them into the right group. For those which is not in the selected 19 popular licenses, we manually classify them into a special group called unknown license.

In this way, we succeed in building the license dictionary including 19 popular licenses, with which we can transform all kinds of style of writing of a license into a normal one.

2. **Build the software evolutionary dataset**

   OSS software in user-contributed OSS ecosystem usually evolves frequently. With the evolution of software, source code, license, and dependencies are also changing. However, an OSS software does not always reuse the latest version of its dependencies, thus deciding the proper version of dependencies becomes important. To accelerate the detection, we build the software evolutionary dataset, in which the licenses and dependencies with licenses of all versions of every package in `npm` are recorded.

   We first use the public API[12] of `npm` to get the metadata of all versions of packages. Then for licenses, we normalize them according to the license dictionary we built before. For dependencies, we list all dependencies and decide the proper version and its license for each dependency. Note that `npm` use semantic versioning standard to manage versions. The first release should start from 1.0.1. After this, changes should be handled according to Table 2[13]. Developers can choose versions for the dependencies in some rules. Table 3 shows some examples for choosing version 1.0.1 for a dependency.

[12] `https://www.npmjs.com/-/all`

[13] `https://docs.npmjs.com/getting-started/semantic-versioning`

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: Examples of how to specify the ranges.

TYPE	**EXAMPLE**
Patch releases	1.0 or 1.0.x or 1.0.1
Minor releases	1 or 1.x or $\hat{1}$.0.1
Major releases	* or x

After we get normalized license and dependencies with the proper version of all versions of a package, we can build software evolutionary dataset for packages in `npm`. The software evolutionary dataset includes information of versions, licenses, and dependencies.

3. **Build license compatibility network**

   To specify the compatibility between two licenses, a license compatibility network is built. Our license compatibility network is based on license compatibility network created by David A. Wheeler[14] which is shown in Figure 3. Each arrow denotes a one-directional compatibility. But to detect the most serious license compliance risk, we assume that the four permissive licenses, including Public Domain license, MIT/X11 license, ISC license, BSD family license and Apache-2.0 license, are compatible with each other and have no license compliance risk. Our detection only detects the incompatible relationship in this license compatibility network.

4. **Detect direct and indirect dependency risk**

[14] `https://www.dwheeler.com/essays/floss-license-slide.html`

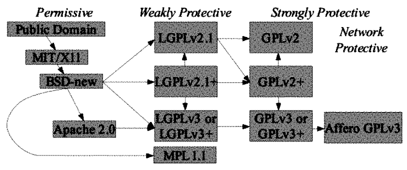


Figure 3: License compatibility network.

Since we have built the software evolutionary dataset and the license compatibility network, detecting direct and indirect dependency risk becomes possible. For a package in `npm`, we first build a dependency chain tree for it. Every direct or indirect dependency in this dependency chain tree is attached with its license. Then we detect if the license of this package is compatible with the licenses of direct and indirect dependencies in this dependency chain tree. At last, we report the license compliance risk we found in the detection. To accelerate this step, we also use some strategies such as preliminary screening the packages having a high possibility of having risks and select licenses for packages which never change its license in advance.

5. **Detect self risk**

   In this detection, we want to detect if a license of a package is not compatible with one source code file of this package. To do this detection, we rely on `npm` to install the detected package. By this way, we get the actual source code when we reuse the detected package. Note that we also get the needed source code of direct and indirect dependencies at the same time. Then we adopted ninka to detect the license of source files. Ninka is a sentence-based license detection tool which can identify 110 different licenses with 93% accuracy in a quite short time [4]. Note that ninka reports UNKNOWN in case that a license is found but not recognized. Another special result None is reported in case that the source file has no license. The style of writing of license reported by ninka is different from our normalized one, so we do a transformation and detect if the license of this package is compatible with the licenses of all source code files of this package. At last, we report the license compliance risk we found in the detection.

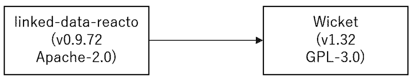


Figure 4: A dependency chain of `linked-data-reactor` package.

3.4 Examples

In this section, we show some examples of packages detected as having license compliance risks in the detection.

1. **Direct dependency risk**:

   `linked-data-reactor`[15] package is detected as having direct dependency risk. Figure 4 shows a dependency chain in the dependency tree of this package. The latest version of `linked-data-reactor` package is 0.9.72, released on March 31, 2017. This version of `linked-data-reactor` depends on the latest version of a package called `wicket`[16]. The latest version of `wicket` is 1.3.2, released on June 4, 2016. Since the version 0.9.72 of `linked-data-reactor` package is licensed under Apache-2.0 license while the latest version of `wicket` is licensed under GPL-3.0 license, a direct dependency risk is detected because of the incompatibility between Apache-2.0 license and GPL-3.0 license.

2. **Indirect dependency risk**:

   `mucbuc-filebase`[17] package is detected as having direct dependency risk. Figure 5 shows a dependency chain in the dependency tree of this package. The latest version of `mucbuc-filebase` package is 0.0.4, released on April 31, 2017. This version of `mucbuc-filebase` depends on the latest version of a package called `walk-json`[18]. The latest version of `walk-json` is 0.0.2, released on January 22, 2017. `mucbuc-filebase` package is licensed under ISC license while `walk-json` is licensed under MIT license. ISC license and MIT license are compatible with each other. But `walk-json` depends on a package called `traverjs`[19]. The depended version of `traverjs` package is licensed under GPL-2.0 license, with which both

[15] `https://www.npmjs.com/package/linked-data-reactor`
[16] `https://www.npmjs.com/package/wicket`
[17] `https://www.npmjs.com/package/mucbuc-filebase`
[18] `https://www.npmjs.com/package/walk-json`
[19] `https://www.npmjs.com/package/traverjs`

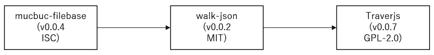


Figure 5: A dependency chain of `mucbuc-filebase` package.

`walk-json` package and `walk-json` package are not compatible. For `walk-json` package, `traverjs` package will cause a direct dependency risk. While for `mucbuc-filebase`, `traverjs` package will cause an indirect dependency risk.

3. **Self risk**:

   An example of packages having self risk is `ncmb-cli`[20] package. `ncmb-cli` package is licensed under MIT license. When a developer downloads this package with `npm`, a source code file called sqlparser.pegjs can be found in the directory $/ncmb - cli/node_{m}odules/node - sqlparser/peg/$. Ninka identifies the license of this file as GPL-2.0 license. Since GPL-2.0 license is not compatible with MIT license, a self risk is detected.

[20]`https://www.npmjs.com/package/ncmb-cli`

4 Empirical Study on License Compliance Risk in `npm`

OSS ecosystems consist of software projects that are developed and evolve together in a shared environment [10]. Among all kinds of OSS software ecosystems, user-contributed OSS ecosystem is the one of which software projects are contributed by users. With the rise of user-contributed OSS ecosystem, reuse of abundant OSS software becomes more and more popular. At the same time, the risks caused by license compliance have become a serious issue. In this section, we set `npm` as our target to do a large-scale empirical study to understand the prevalence of license compliance risk. `npm` is the largest software ecosystem, hosting over 450,000 JavaScript packages. Millions of packages are installed from the `npm` repository on an everyday basis. The popularity of `npm` makes `npm` a good target to conduct our empirical study.

4.1 Research Question

To understand the prevalence of license compliance risk, we want to know what type of common license compliance risks are in `npm` and how they appear. Based on these questions, we set our research question as follows:

- **RQ1** *What is the proportion of packages with license compliance risk in* `npm`*?*
- **RQ2** *Is the reuse of packages licensed under the copyleft license more likely to cause license compliance risk?*
- **RQ3** *Does transitive dependency have an impact on the occurrence of license compliance risk?*
- **RQ4** *What are the characteristics of license compliance risk at file level?*

4.2 Results

To answer RQ1, we first apply our method of detecting direct and indirect dependency risks to all packages we collected in `npm`. As a result, only 2,704 packages are detected as having direct or indirect dependency risk out of 419,708 packages. The proportion is only 0.644%, which is very different from what we have expected. To ascertain the reason, we count the proportion of the selected licenses in `npm`. Figure 6 and Table 4 show the result.

The result shows that the permissive licenses take a large part of all licenses while the copyleft licenses are not widely used in `npm`. It suggests that the proportion of permissive

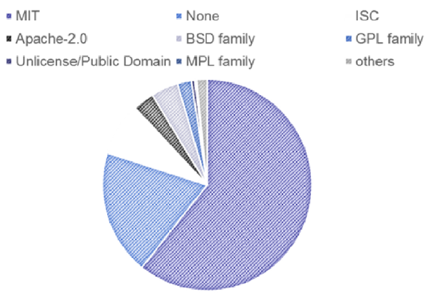


Figure 6: The proportion of the selected licenses in `npm`.

Table 4: The proportion of the selected licenses in `npm`.

License	**Proportion**
MIT	60.75%
None	19.14%
ISC	8.06%
Apache-2.0	3.24%
BSD family	4.24%
GPL family	2.18%
Unlicense/PublicDomain	0.50%
MPL family	0.27%
Other	1.62%
All	100%

and copyleft license have an impact on the prevalence of license compliance risk in OSS ecosystem.

To answer RQ2, we selected some copyleft licenses as the target. The selected copyleft licenses are listed as follows:

```
GPL-2.0 (GNU General Public License version 2) and GPL-2.0+
GPL-3.0 (GNU General Public License version 3) and GPL-3.0+
LGPL-2.1 (GNU Lesser General Public License version 2.1) and LGPL-2.1+
LGPL-3.0 (GNU Lesser General Public License version 3.0) and LGPL-3.0+
AGPL-3.0 (GNU Affero General Public License version 3)
```

Then we collected the packages including a package licensed under the selected copyleft licenses in its dependency chain. As a result, we collected 4,067 packages. Among them, 2,704 packages are detected as having direct or indirect dependency risk. Note that all

Table 5: The statistic of the depth the direct or indirect dependency risk happened.

Depth	**#Risks**	**#All**	**Probability**
Depth1	2,115	20,079	0.1050
Depth2	1,052	57,109	0.0184
Depth3	352	79,065	0.0044
Depth4	82	73,468	0.0011
Depth5	19	60,442	0.0003
Depth6	2	45,410	0.00004

packages detected as having direct or indirect dependency risk in the first detection are included in these 4,067 packages. The proportion of packages detected as having the risk is 66.84%. The high proportion suggests that the reuse of packages licensed under the copyleft license is more likely to cause license compliance risk. One possible explanation is that the developers simply ignore the license compliance risk when they reuse a package.

To answer RQ3, we analyze the depth the direct or indirect dependency risk happened. For example, the direct dependency risk is denoted as depth 1. For indirect dependency risk, the depth is increased. We also count the number of all depths of the direct or indirect dependencies to calculate the probability of the direct or indirect dependency risk happened in every depth. Table 5 shows the result.

The probability of the direct or indirect dependency risk happened is decreased with the dependency depth increasing. The result suggests that the direct or indirect dependency risk has a tendency to happen in the shallow dependency.

To answer RQ4, we select the 2,704 packages detected as having direct or indirect dependency risk in the first detection to conduct the self risk detection since we assume that the packages having direct or indirect dependency risk have a high possibility of having self risk as well. As a comparison, we also randomly select 2,000 packages with no direct or indirect dependency risk to conduct the self risk detection. As a result, 964 packages in 2,704 packages are detected as having self risk. The proportion is 35.18%, while none of the 2,000 safe packages are detected as having the risk. However, in the 9,679,468 source code files of 2,704 packages, only 291,340 files is detected as not compatible with the packages. The proportion is only 0.03%. The results prove our assumption that the packages having direct or indirect dependency risk have a high possibility of having self risk as well. Meanwhile, the source code files causing compliance risk only take a small part of all source code files of a package.

4.3 Answering RQs

Revisiting the research questions:

- **RQ1:** *What is the proportion of packages with license compliance risk in* `npm`*?* The proportion of packages with license compliance risk in `npm` is only 0.644%. The reason is that the proportion of permissive and copyleft license have an impact on the prevalence of license compliance risk in OSS ecosystem.
- **RQ2:** *Is reuse of packages licensed under the copyleft license more likely to cause license compliance risk?* Yes, reuse of packages licensed under the copyleft license is more likely to cause license compliance risk. Developers should pay more attention to packages licensed under the copyleft license if they want to reuse these packages.
- **RQ3:** *Does transitive dependency have an impact on the occurrence of license compliance risk?* Yes, it does. The direct or indirect dependency risk has a tendency to happen in the shallow transitive dependency.
- **RQ4:** *What are the characteristics of license compliance risk in file level?* The packages having direct or indirect dependency risk have a high possibility of having self risk as well. Meanwhile, the source code files causing compliance risk only take a small part of all source code files of a package.

5 Copyright Inconsistency Risk

5.1 Background

Software copyright is used to prevents the unauthorized copying of software. It is usually written in the header part of source code. Here is an example of a license statement taken from getopt.c file in Linux kernel:

```
[...]
// SPDX-License-Identifier: GPL-2.0
/*
 * arch/alpha/boot/bootp.c
 *
 * Copyright (C) 1997 Jay Estabrook
 *
 * This file is used for creating a bootp file for the Linux/AXP kernel
 *
 * based significantly on the arch/alpha/boot/main.c of Linus Torvalds
 */
 [...]
```

The copyright is Copyright (C) 1997 Jay Estabrook. In this copyright declaration, Jay Estabrook is the copyright holder. But Jay Estabrook is not the only contributor to this file. Actually, this file is contributed by 6 contributors. Here comes a problem: who is the real copyright holder of this file? Could any contributor be a copyright holder even they just contribute a few lines of code? Could a developer claim him as a copyright holder even that his copyright information is not included in the header part of source code?

It's important for developers or companies reusing the OSS software to know the real copyright holder of OSS software. Recent years some contributors of Linux kernel complained some companies for the violation of copyright. But these companies need to judge if these contributors have the right to complain them. The unclear definition of copyright holder is a big potential risk for the developers or companies reusing the OSS software. A typical example is the inconsistency between the declared copyright holders and the main contributors. This makes the developers or companies not able to identify the real copyright holders. In this study, we conduct an empirical study on Linux kernel to understand the inconsistency between the declared copyright holders and the main contributors.

5.2 Definition

For the purpose of this study, a copyright inconsistency risk refers to the situation when the declared copyright holders of OSS software are not the main contributors.

5.3 Example

In this section, we show some examples of files with copyright inconsistency risk in Linux kernel.

The file sama5.c is an example. The declared copyright holder is shown in the following:

```
[...]
/*
 *  Setup code for SAMA5
 *
 *  Copyright (C) 2013 Atmel,
 *                2013 Ludovic Desroches <ludovic.desroches@atmel.com>
 *
 * Licensed under GPLv2 or later.
 */
[...]
```

But the developer who contributed the largest proportion of this file is Alexandre Belloni. So the declared copyright holder is different with the main contributors.

Another example is coda.h. The declared copyright holder is shown in the following:

```
[...]
/*
 *
 * Based on cfs.h from Mach, but revamped for increased simplicity.
 * Linux modifications by
 * Peter Braam, Aug 1996
 */
[...]
```

While the main contributor is Linus Torvalds who is not the declared copyright holders.

Another example is where no copyright holder is recorded in the header part. brcmphy.h is one of this kind of example. The header part of this file is shown as follows:

```
[...]
/* SPDX-License-Identifier: GPL-2.0 */
[...]
/* All Broadcom Ethernet switches have a pseudo-PHY at address 30 which is used
 * to configure the switch internal registers via MDIO accesses.
 */
[...]
```

The main contributor of this file is Florian Fainelli, who is not written in the header part.

The above three files are examples with copyright inconsistency risk.

6 Detection of Copyright Inconsistency Risk

6.1 Method

Our method is divided into 3 steps:

1. **Build the header comments dataset** We build a comment extractor and use it to extract the comments from files in Linux kernel. We build the dataset of the comments by using file name as an index. In this dataset, not only the file name but also the path of the file is recorded. And for the comment extractor, we extract the comments based on the rules of the beginning and the end of the comments in C/C++ programming language. After extraction, we clear the extracted comments and only store the real comments. For example, a part of the comments of a file named sqlparser.pegjs in the directory sqlparser.pegjs is shown as follows:

   ```
   [...]
   /* AFS cell and server record management
    *
    * Copyright (C) 2002 Red Hat, Inc. All Rights Reserved.
    * Written by David Howells (dhowells@redhat.com)
    *
   [...]
   ```

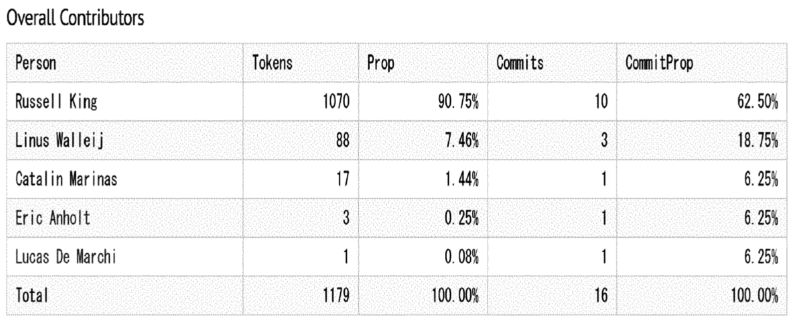
   As a result, the comments are stored as follows:

   ```
   [...]
   AFS cell and server record management
   Copyright (C) 2002 Red Hat, Inc. All Rights Reserved.
   Written by David Howells (dhowells@redhat.com)
   [...]
   ```

2. **Build the contributor dataset** We use `cregit`[21] to collect the main contributors of files. `cregit` can collect information of contributors to the Linux kernel, such as name, contributed tokens, and the proportion of contributed tokens in all tokens etc. `cregit` provide all information in an HTML file. An example is a file named clcd.h in the directory */4.14/include/linux/amba/*. The information of contributors is listed in a table as shown in Figure 7.

   We build a web crawler to extract the data from HTML file created by `cregit`. The data is also organized with a table in our dataset.

[21] `https://github.com/cregit/cregit/`


Overall Contributors

Person	Tokens	Prop	Commits	CommitProp
Russell King	1070	90.75%	10	62.50%
Linus Walleij	88	7.46%	3	18.75%
Catalin Marinas	17	1.44%	1	6.25%
Eric Anholt	3	0.25%	1	6.25%
Lucas De Marchi	1	0.08%	1	6.25%
Total	1179	100.00%	16	100.00%

Figure 7: The information of contributors collected by `cregit`.

In this step, we treat contributors who contributed the most tokens as main contributors.

3. **Detect if the main contributor is in the header comments** With the built header comments dataset and contributor dataset, we detect if the main contributor is in the header comments by matching the name of main contributors with all lines of comments. If the name is not found in any line of the comments, we detect this file as having copyright inconsistency risk.

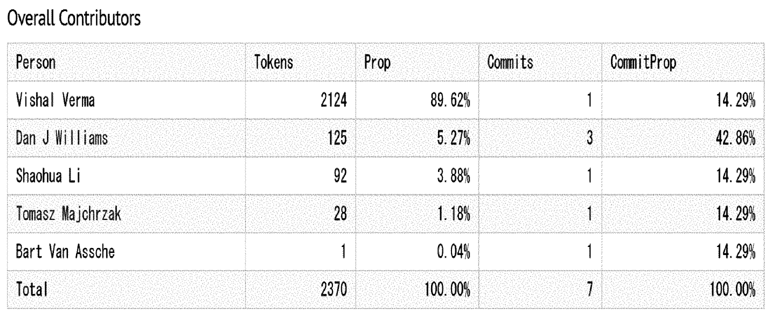
   A file named badblocks.c in the directory */4.14/block/* shows an example that the main contributor in the header comments is matched by a comment line. The comments in the header are:

```
[...]
/*
 * Bad block management
 *
 * - Heavily based on MD badblocks code from Neil Brown
 *
 * Copyright (c) 2015, Intel Corporation.
 *
[...]
```

   While the information extracted by `cregit` is shown in Figure 8.

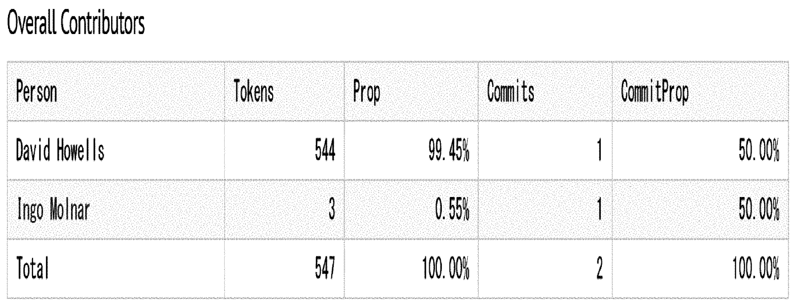
   It's easy to know that no comments in the header can be matched by the main contributor named Vishal Verma.

   While a file named a.out-core.h in the directory */arch/alpha/include/asm/* shows an example that the main contributors do not match any line of the comments. The comments in the header are:


Overall Contributors

Person	Tokens	Prop	Commits	CommitProp
Vishal Verma	2124	89.62%	1	14.29%
Dan J Williams	125	5.27%	3	42.86%
Shaohua Li	92	3.88%	1	14.29%
Tomasz Majchrzak	28	1.18%	1	14.29%
Bart Van Assche	1	0.04%	1	14.29%
Total	2370	100.00%	7	100.00%

Figure 8: The information of contributors collected by `cregit`.


Overall Contributors

Person	Tokens	Prop	Commits	CommitProp
David Howells	544	99.45%	1	50.00%
Ingo Molnar	3	0.55%	1	50.00%
Total	547	100.00%	2	100.00%

Figure 9: The information of contributors collected by `cregit`.

```
[...]
/* a.out coredump register dumper
 *
 * Copyright (C) 2007 Red Hat, Inc. All Rights Reserved.
 * Written by David Howells (dhowells@redhat.com)
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public Licence
 * as published by the Free Software Foundation; either version
 * 2 of the Licence, or (at your option) any later version.
 */
[...]
```

While the information extracted by `cregit` is shown in Figure 9.

a.out-core.h file is detected as having no copyright inconsistency risk.

6.2 Threats to Validity

In this study, we proposed a method of detecting the copyright inconsistency risk in Linux kernel. But there are still some threats to the validity of the result. Firstly, Linux kernel has a long history, the early contributors are untraceable before the source code files are uploaded to GitHub by Linux kernel. Another threat is some contributors only recorded the information of their companies or nickname in the copyright. This kind of cases is detected as having copyright inconsistency risk since it is impossible for us to find it out.

7 Empirical Study on Copyright Inconsistency Risk in Linux kernel

To understand the prevalence of copyright inconsistency risk and how they happened, we conduct an empirical study on Linux kernel. The version of Linux kernel we selected is 4.14. We download it from GitHub[22].

7.1 Research Question

To achieve our purpose, we set our research question as follows:

- **RQ** *What is the proportion of files with copyright inconsistency risk in Linux kernel?*

7.2 Result

We apply our method to all files in Linux kernel version 4.14. As a result, In 45,471 files of Linux kernel about 14,304 files have the copyright inconsistency risk. The proportion is 31.46%. The result suggests that the copyright inconsistency risk is very common in Linux kernel. Then we manually analyze the evolutionary history of some files detected as having copyright inconsistency risk and find some cases how copyright inconsistency risk happened:

- Linux kernel has a long history. Before Linux kernel is uploaded to GitHub, many contributors contributed source codes. The information of these early contributors can't be traced.
- Some contributors are the employee of companies, they only recorded the information of their companies instead of themselves.
- As time pass by, some contributors continually contribute source codes and become the main contributor, but they are not the original creators and don't write their copyright information in the header.

The first case and second case are copyright inconsistency risks caused by the internal factor, while the third case is caused by the external factor. In other words, different with the first and the second case, the third one is able to be prevented by the contributors of the files. To ascertain the prevalence of this preventable copyright inconsistency risks, we conduct an evolutionary study on 11,873 files in the directory */arch* of Linux kernel. As

[22]`https://github.com/torvalds/linux`

a result, in 7,603 files detected as having copyright inconsistency risk, only 315 files are the third case. The result suggests the difficulties of knowing the real copyright holders of files in Linux kernel.

7.3 Answering the RQ

Revisiting the research question:

- **RQ** *What is the proportion of files with copyright inconsistency risk in Linux kernel?*

  The proportion of files with copyright inconsistency risk in Linux kernel is 31.46%. The copyright inconsistency risk is very common in Linux kernel.

8 Related Work

8.1 Software License

Many studies in software engineering investigated software license. There are some studies that devote to identify licenses [4, 8, 13]. Based on these studies, some researchers analyzed software licenses in open source projects and revealed some license issues. Di Penta et al. [3] 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. [6] proposed a method to understand licensing compatibility issues in software packages. They mainly focused on the compatibility between license declared in packages and those in source files. In another research by Di Penta et al. [7], 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. Alspaugh et al. [1] proposed an approach for calculating conflicts between licenses in terms of their conditions. Wu et al. proposed an approach to find license inconsistencies in similar files [18]. 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. Vendome et al. [16] performed a large empirical study of Java applications and found that changing license is a common event and 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 adopt 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 tool support in guiding developers in choosing and changing licenses and in keeping track of the rationale of license changes [15].

8.2 License Compliance

License compliance is an import 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 [19]. 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 [5]. Van et al. proposed an approach that can uncover license compliance inconsistencies by analyzing the Concrete Build Dependency Graph of a software system [14]. Vendome et al. studied the rationale of developers in choosing and changing licenses and investigated the problem of traceability of license changes [17]. They provided a vision of ensuring license compliance of a system. Kapitsaki et al. proposed an approach of automating the license compliance with a process that examines the structure of Software Packages Data Exchange [9].

9 Conclusion and future work

In this paper, we proposed an approach to detect the license compliance risk of software projects in OSS ecosystems, which can expose direct and indirect dependencies of which the licenses are incompatible with the license of detected software projects.

With this method, we conducted an empirical study to understand the current situation of license compliance risk in `npm`, a popular user-contributed OSS ecosystem. We discovered that the proportion of packages detected as having direct or indirect dependency risk is only 0.644%. The reason is that permissive licenses take a large proportion of all licenses while the copyleft licenses are not widely used in `npm`. We also revealed that the reuse of packages licensed under the copyleft license is more likely to cause license compliance risk. The result of the empirical study also suggests that the direct or indirect dependency risk has a tendency to happen in the shallow dependency. We also discovered that the packages having direct or indirect dependency risk have a high possibility of having self risk as well. Meanwhile, the source code files causing license compliance risk only take a small proportion of all source code files of a package.

In this study, we also proposed a method to detect the copyright inconsistency risk. With this method, we conducted an empirical study on Linux kernel to understand the prevalence of copyright inconsistency risk and how they happened. As a result, the proportion of source code files having the copyright inconsistency risk is 31.46%. With a manual analysis of some source code files having the copyright inconsistency risk, we discovered that there are some cases how copyright inconsistency risk happened: the early version of Linux kernel is untraceable; some contributors recorded their companies instead of their names; some contributors not included in the declared copyright holders became the main contributor with the time passing by.

The results of this study proved the prevalence of license compliance risk and copyright inconsistency risk in user-contributed OSS ecosystem. When developers reuse OSS software, they should pay special attention to open source license and software copyright to prevent themselves from legal risks.

For future work, we will apply our method to other user-contributed OSS ecosystems such as the CRAN package repository[23] and RubyGems [24]. It will be interesting to compare the situations of license compliance risk and copyright inconsistency risk in different

[23]`https://cran.r-project.org/`

[24]`https://rubygems.org/`

OSS ecosystems. We will also improve our method of detecting license compliance risk and copyright inconsistency risk and provide them with web services.

謝辞

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Katsuro Inoue for giving me the opportunity to study in Japan and work with him. Without his continuous support and help, it would not have been possible for me to complete my thesis.

Besides my supervisor, I would like to thank the rest of the members in my research group: Prof. Daniel M. German, Prof. Yuki Manabe, Prof. Takashi Ishio, Prof. Raula Gaikovina Kula and Yuhao Wu, for their valuable comments and suggestions in my research.

I would also like to thank the rest of the members of Inoue laboratory, especially Mrs. Mizuho Karube. With them, I left a lot of good memories during the times in Japan.

Last but not least, I am very grateful to my parents. They support my study and living in Japan and pull me through many difficulties.

参考文献

[1] T.A. Alspaugh, H.U. Asuncion, and W. Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. In *Proceedings of the 17th International Requirements Engineering Conference (RE2009)*, pp. 24–33, 2009.

[2] Barry W. Boehm. Improving software productivity. *Computer*, Vol. 20, No. 9, pp. 43–57, September 1987.

[3] Massimiliano Di Penta, Daniel M. German, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE2010)*, pp. 145–154, 2010.

[4] Daniel M German, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the 25th International Conference on Automated Software Engineering (ASE2010)*, pp. 437–446, 2010.

[5] Daniel German and Massimiliano Di Penta. A method for open source license compliance of java applications. *IEEE software*, Vol. 29, No. 3, pp. 58–63, 2012.

[6] D.M. German, M. Di Penta, and J. Davies. Understanding and auditing the licensing of open source software distributions. In *Proceedings of the 18th International Conference on Program Comprehension (ICPC2010)*, pp. 84–93, 2010.

[7] D.M. German, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *Proceedings of the 6th Working Conference on Mining Software Repositories (MSR2009)*, pp. 81–90, 2009.

[8] Robert Gobeille. The FOSSology project. In *Proceedings of the 5th Working Conference on Mining Software Repositories (MSR2008)*, pp. 47–50, 2008.

[9] Georgia M Kapitsaki, Frederik Kramer, and Nikolaos D Tselikas. Automating the license compatibility process in open source software with spdx. *Journal of Systems and Software*, Vol. 131, pp. 386–401, 2017.

[10] Mircea Lungu, Michele Lanza, Tudor Grba, and Romain Robbes. The small project observatory: Visualizing software ecosystems. *Science of Computer Programming*, Vol. 75, No. 4, pp. 264 – 275, 2010. Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008).

[11] M Douglas McIlroy, JM Buxton, Peter Naur, and Brian Randell. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering (ICSE1968)*, pp. 88–98, 1968.

[12] Thomas A. Standish. An essay on software reuse. *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, pp. 494–497, Sept 1984.

[13] Timo Tuunanen, Jussi Koskinen, and Tommi Kärkkäinen. Automated software license analysis. *Automated Software Engineering*, Vol. 16, No. 3-4, pp. 455–490, 2009.

[14] Sander Van Der Burg, Eelco Dolstra, Shane McIntosh, Julius Davies, Daniel M German, and Armijn Hemel. Tracing software build processes to uncover license compliance inconsistencies. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pp. 731–742. ACM, 2014.

[15] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. License usage and changes: a large-scale study on github. *Empirical Software Engineering*, Vol. 22, No. 3, pp. 1537–1577, 2017.

[16] Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel M. Germán, and Denys Poshyvanyk. License usage and changes: A large-scale study of java projects on github. In *The 23rd IEEE International Conference on Program Comprehension, ICPC 2015*, 2015.

[17] Christopher Vendome and Denys Poshyvanyk. Assisting developers with license compliance. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 811–814. ACM, 2016.

[18] Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M German, and Katsuro Inoue. A method to detect license inconsistencies in large-scale open source projects. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR2015)*, pp. 324–333, 2015.

[19] Hongyu Zhang, Bei Shi, and Lu Zhang. Automatic checking of license compliance. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pp. 1–3. IEEE, 2010.