How Are Developers Treating License Inconsistency Issues? A Case Study on License Inconsistency Evolution in FOSS Projects

Yuhao Wu[1], Yuki Manabe[2], Daniel M. German[3], and Katsuro Inoue[1]

[1] Graduate School of Information Science and Technology,
Osaka University, Japan
{wuyuhao,inoue}@ist.osaka-u.ac.jp
[2] Graduate school of Science and Technology,
Kumamoto University, Japan
y-manabe@cs.kumamoto-u.ac.jp
[3] Department of Computer Science,
University of Victoria, Canada
dmg@uvic.ca

Abstract. A license inconsistency is the presence of two or more source files that evolved from the same original file containing different licenses. In our previous study, we have shown that license inconsistencies do exist in open source projects and may lead to potential license violation problems. In this study, we try to find out whether the issues of license inconsistencies are properly solved by analyzing two versions of a FOSS distribution—Debian—and investigate the evolution patterns of license inconsistencies. Findings are: license inconsistencies occur mostly because the original copyright owner updated the license while the reusers were still using the old version of the source files with the old license; most license inconsistencies would disappear when the reusers synchronize their project from the upstream, while some would exist permanently if reusers decide not to synchronize anymore. Legally suspicious cases have not been found yet in those Debian distributions.

Keywords: software license; code clone; license inconsistency

1 Introduction

Free and open source software (FOSS) is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone. FOSS should always be distributed under certain open source software licenses, otherwise they cannot be reused by others. The license of a source file usually resides in its header comment.

Many studies in software engineering have been done on software license. Some approaches for software license identification have been proposed [3], [6], [8]. Vendome et al. investigated the reasons on when and why developers adopt and change licenses by conducting a survey with the relevant developers [10].

We have developed a method to detect *license inconsistencies* in large-scale FOSS projects [11]. A license inconsistency is defined as two or more source files that evolved from the same original file containing different licenses. For example, file `A` and file `B` have the same program code, but the header of `A` includes GPL-2.0 and `B` includes Apache-2.0[4]. We have applied our detection method to Debian 7.5[5], one of the Linux distributions, and discovered different reasons that caused license inconsistencies and determined whether they are potentially illegal or not.

Although this work has provided us clues that some license inconsistencies indicate potential legal issues, this one-time analysis is not enough to understand how developers are dealing with the issues: whether developers are putting their efforts on eliminating license inconsistencies or they simply ignore the issues. The investigation of the evolution of license inconsistencies can help us understand whether the legal issues of license inconsistency discovered in the previous work are handled by developers. Therefore, we conducted this research and try to answer the following questions:

- **RQ1.** *What are the evolution patterns of license inconsistencies and the underlying reasons?* Analyzing the evolution patterns of license inconsistencies might gives us insight on the reasons that caused them to appear and disappear. The findings are: license inconsistencies appear, persist and disappear due to different reasons. They appear mostly because the original author updates the license while the reusers still use the old version of the files; they persist mostly because the downstream project is not synchronized with the upstream project yet; and they disappear when the downstream project is synchronized with the upstream project.
- **RQ2.** *Is the issue of license inconsistencies properly handled by developers?* The findings are: license inconsistencies are mainly caused by distribution latency, and they will disappear when the developers synchronize their projects from the upstream projects. They persist because the reusers are still using an old version of the files and do not perform the synchronization. We do not consider this as a legal issue.

In order to address these questions, we apply our license inconsistency detection method to Debian 8.2 (which was the newest version when we started this study) in addition to Debian 7.5, and investigate the evolution of license inconsistencies between theses two versions.

The rest of this paper is organized as follows: Section 2 introduces the methodology we employ to address our research questions. Section 3 describes the result we got from our analysis. A discussion of this result is given in Section 4. Section 5 introduces the related work. Finally, we conclude our paper in Section 6.

[4] Note that pairs of no license and any license, and different license versions are reported as license inconsistency.

[5] `https://www.debian.org/`

2 Methodology

In this section we describe how we design our study.

2.1 Obtain license inconsistency groups for Debian 7.5 and Debian 8.2

A *license inconsistency group* is a group of multiple files that have same code contents but with different licenses. As introduced in our previous paper [11], we use the following steps to reveal license inconsistency groups:

1. **Create groups of file clones:** For all the source files in the target projects, we apply `CCFinder` [7] to extract the normalized token sequences of each file. The normalized token sequences is a token sequence of the source code, removing comments and whitespaces and changing all user-defined identifiers to a special token. Note that, although `CCFinder` itself is a clone detection tool, we do not utilize the full functionality of `CCFinder` and we only use it to generate the normalized token sequences of source files. By computing and categorizing the hash value of these token sequences, we then create a group for files that have the same normalized token sequences. We call them *license inconsistency groups*, or *group* for short in the rest of this paper. Each group contains at least two different files; i.e., a unique file is not contained in any group.
2. **Identify licenses for files in each group:** For each group of file clones, `Ninka` [3] is used to identify the license(s) of each file. `Ninka` identifies the license sentences in the comment parts of each file, and compares those with its license database. It can identify more than 110 different OSS licenses and their different versions with 93% accuracy. Meanwhile, it will report "NONE" if the file has no license and "UNKNOWN" if the license sentence of the file does not match the database. The result is a list of licenses for each file group.
3. **Report groups that contain a license inconsistency and calculate the inconsistency metrics:** We compare the license list of each file group. File groups are reported to have license inconsistencies unless all the licenses on the list are exactly the same. The result is a list of file groups that contain one or more types of license inconsistencies.

We apply these steps to Debian 7.5 and 8.2 respectively and obtain the file groups that contain license inconsistencies.

2.2 Compare the difference of groups

All the files in one group have the same token sequences, from which we calculate the hash value as the id for that group. We then compare the id of the groups in each version and get the set of groups that exist: *a*) only in Debian 7.5; *b*) only in Debian 8.2; *c*) in both versions.

The result of this step would be three sets of file groups.

Table 1: Number of packages and files in Debian 7.5 and 8.2.

Number of	Debian 7.5	Debian 8.2
Source Packages	17,160	20,577
Total files	6,136,637	13,124,700
.c files	472,861	767,006
.cpp files	224,267	335,269
.java files	365,213	477,154

Table 2: License inconsistency groups between two versions on Debian.

Number of groups	Debian 7.5	Debian 8.2
Intersection[i]	4062	4062
Relative complement[ii]	2701	2947
Total	6763	7009

[i] Groups that exist in both versions.
[ii] Groups that exist in either version only.

2.3 Investigate the groups manually

The groups that only exist in Debian 7.5 imply that they are eliminated from the new version; those that only exist in Debian 8.2 imply that new license inconsistency cases appears in this new version; those that exist in both versions indicate that they remain during these two versions. We manually examine the different groups and present the results in the next section.

3 Results

The number of packages and files in Debian 7.5 and 8.2 are shown in Table 1. We only consider `.c`, `.cpp` and `.java` files which are the supported types of our detection method. The detection result of license inconsistency groups in Debian 7.5 and 8.2 is shown in Table 2. In this table, intersection means that both two versions of Debian contain that group of license inconsistency; relative complement means only that version of Debian contain that group of license inconsistency. Thus there are 4062 groups of license inconsistencies detected in both versions; 2701 groups only in Debian 7.5; and 2947 groups only in Debian 8.2. By examining the groups that are only in Debian 7.5 we can find out how and why license inconsistencies disappeared in Debian 8.2; while examining the groups that are only in Debian 8.2 we can understand how and why license inconsistencies appeared. The intersection part indicates that these groups of license inconsistencies persisted in Debian 8.2.

3.1 Why do license inconsistencies appear?

There are several reasons that license inconsistencies appear. The key point is that there are multiple copies of the same source file with different licenses in

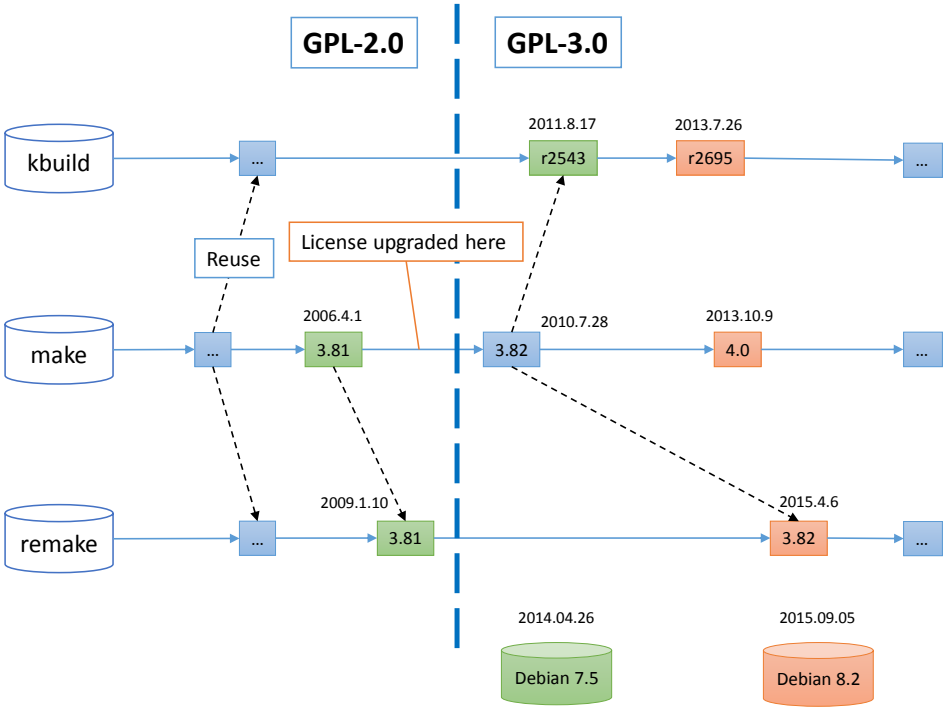

Fig. 1: A case of license inconsistency evolution which involves project `make`, `remake` and `kbuild`.

the same distribution. We describe the general reasons that license inconsistency appears, thus the examples given in this section are not limited to those that appear in Debian 8.2. We categorize them into three types according to the reason that caused the copy of files.

– **Internal copy-and-paste of source files but their licenses are different.** If copies of the same file exist in one project (we call these internal copies), they should also exist in the final distribution (e.g. Debian 7.5). Thus a case of license inconsistency will be reported by our method if they contain different licenses.

For example, in a project named `FreeMedForms`, some source files in a `plugins` directory are under BSD3 license. These files are copied to other directories with their licenses changed to GPL-3.0+.

This type of license inconsistency could exist for just a short time, if the difference of license was due to mistakes and was later on fixed by developers. On the other hand, if developers decided to distribute these source files under different licenses, it would exist permanently.

– **Different versions of the same project are included in the same distribution.** Similar to the previous reason, if different versions of the same

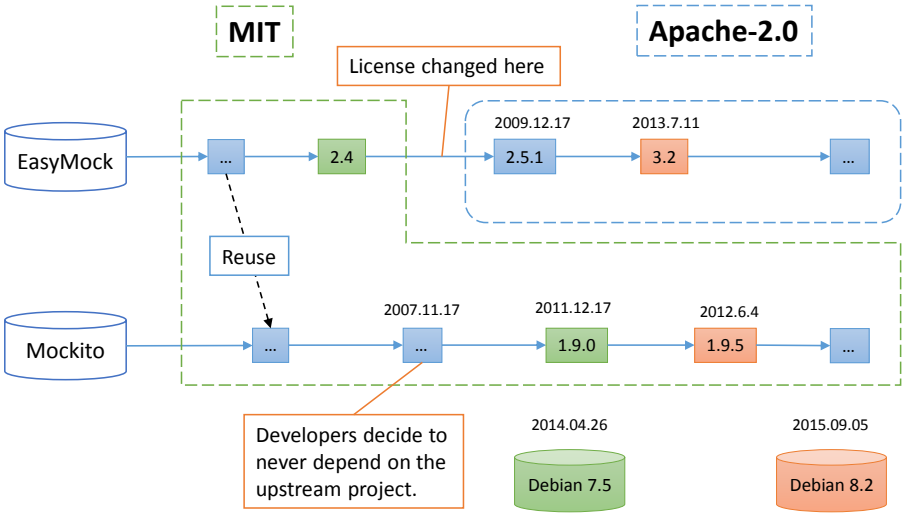

Fig. 2: A case of license inconsistency evolution which involves project `EasyMock` and `Mockito`.

project which causes license inconsistencies are included in the final distribution, those license inconsistencies will be reported by our method.

For example, there is a project named `groovy` which contains files that had no license in version 1.7.2, and were then added a Apache-2.0 license in version 1.8.6. Both of these two versions of this project are included in Debian 7.5, thus this license inconsistency is reported.

– **Upstream and downstream projects both exist in the same distribution.** In this research, if project `B` reuses source files from project `A` by copy-and-paste, we call project `B` the downstream project and project `A` the upstream project. While the previous two reasons are about the same project, this reason involves multiple projects. Files from the upstream project are reused in the downstream project, and the license of these files were changed either by the original author or the reuser. If both of these projects are included in the same distribution, the license inconsistency will be reported by our method.

An example is shown in Figure 1. As we can see from this graph, several source files in `remake` and `kbuild` project were originally from the `make` project, where license upgrade occurred in year 2010. Debian 7.5 includes older versions of `make` and `remake`, where the license was still GPL-2.0, while the newer version of `kbuild` contains the GPL-3.0 license. Thus license inconsistency is reported for this case.

3.2 Why do license inconsistencies persist?

License inconsistencies usually exist in several continuous versions of a distribution. Some of them disappear soon, some exist for a long time, while some even persist forever. From the result in this research, we can summarize them into two reasons.

– **Source files are not yet synchronized, but license inconsistencies will eventually disappear when they are.** This type of license inconsistencies occurred because the downstream project has not yet been synchronized with the upstream project where the license of source files were changed. However, since the developers of the downstream projects are still synchronizing the project from the upstream regularly, this type of license inconsistencies will be eliminated eventually.

For example, in the case of the project `JSON-lib` and `jenkins-json` described earlier, although license inconsistencies appeared in Debian 7.5 (where `jenkins-json` still uses source files under Apache-2.0/MIT dual license), they disappeared in Debian 8.2, where developers of `jenkins-json` project synchronized from the upstream project and the license all become Apache-2.0 only. Though this case of license inconsistency disappeared in Debian 8.2, we consider it as a typical example to explain why license inconsistencies would exist for only a period of time and disappear when the source files are synchronized.

– **Downstream project no longer synchronizes from upstream, and license inconsistencies will likely exist permanently.** In this case, developers of downstream project chose to no longer synchronize from the upstream project, thus the license inconsistencies are likely to exist forever, unless the synchronization resumes.

As shown in Figure 2, among the results we found a project named `Mockito` which copy-and-owned several source files from a project named `EasyMock`. The license of these files in the upstream project were changed from MIT to Apache-2.0 in year 2009, however the `Mockito` project still uses the original MIT license. Besides, `Mockito` project made some changes to the source code of these files by their own, and never again synchronized from `EasyMock`. After checking the history of these files in `Mockito` project, we found that one of the commit in year 2007 contains the following commit message: "*umbilical cord between mockito package and easymock package is cut!*", which implies that they will never synchronize from the upstream project. Thus this case of license inconsistency is likely to exist permanently, unless `Mockito` project decides to synchronize from `EasyMock` again.

3.3 Why do license inconsistencies disappear?

We observed several cases where license inconsistencies disappeared from Debian 8.2. The reasons are summarize as follows.

– **Downstream project synchronized from upstream project.** When the downstream project synchronized from the upstream project, the license of the source files becomes the same, thus the license inconsistencies disappear.

Again, from Figure 1 we can see that Debian 8.2 updated all these three projects to a newer version where all of their licenses are upgraded to GPL-3.0, thus this license inconsistency disappears.

The case of project `JSON-lib` and `jenkins-json` discussed earlier also applies here.

– **The source code that contained the license inconsistency was removed or changed—thus no longer identical.** In this research since we only inspect identical files, only files that contain the same token sequences are considered that they are from the same origin. If the source code of a file changed dramatically which made their token sequences different from the corresponding files, or if the relevant source file was removed in the new version, then the license inconsistencies will no longer be reported by our method.

For example, there is a file in project `icu` which is under IBM copyrights in both versions of Debian. This file was reused in project `openjdk-7` but with a GPL-2.0 license in Debian 7.5. Thus this case of license inconsistency was reported in Debian 7.5. However, the source code of the file in `openjdk-7` was changed in Debian 8.2 while the license remained the same. Our method no longer not consider them as file clones since they have different token sequences, thus the license inconsistency disappears in the newer version of Debian.

4 Discussion

4.1 Revisiting the research questions

The answer to our research questions are as follows:

RQ1: *What are the evolution patterns of license inconsistencies and the underlying reasons?*

- **Appear.** *i*) Internal copy-and-paste of source files in a project but their licenses are different; *ii*) Different versions of the same project are included in the same distribution; *iii*) Upstream and downstream projects both exist in the same distribution.
- **Persist.** *i*) Source files are not yet synchronized, but license inconsistencies will eventually disappear when they are; *ii*) Downstream project no longer synchronize from upstream, and license inconsistencies are likely to persist permanently.
- **Disappear.** *i*) Downstream project synchronized from upstream project; *ii*) Source code which contained license inconsistency was removed or changed and we consider them as different files.

RQ2: *Is the issue of license inconsistency properly handled by developers?* The evolution of license inconsistencies in Debian shows that, they are mainly caused by distribution latency and will be eliminated when the downstream

projects get synchronized. We found license inconsistencies appear mostly because the original author modified the license while the reusers were still using the old version of the file; they disappear because the files are synchronized with the upstream projects; and they persist if the downstream projects were not synchronized.

4.2 Effectiveness of this approach

This approach is effective to analyze how developers are addressing the issues of license inconsistency in a certain software ecosystem. As shown in the results, we could use this method to reveal how license inconsistency groups evolved over time. By analyzing different groups we could have a basic idea on how developers are treating license inconsistency issues. For example, by analyzing the license inconsistency groups that only exists in the old version, we could learn why license inconsistencies disappeared in the newer version and what efforts did the developers put to eliminate them.

We could apply this same approach to other software systems to study how different communities are dealing with license inconsistency issues.

4.3 Threats to validity

Internal validity. We use the same methodology as our previous study to detect license inconsistency cases, which relies on the token sequence generation from `CCFinder` and the license identification from `Ninka`. For file clone detection, we use normalized token sequences as the metric to decide clones. If source files are modified a lot (e.g. add/remove several statements), they might not be recognized as clones. We could use approaches that detect similar source files to mitigate this problem. Regarding license identification, `Ninka` is state-of-the-art license identification tool which has an accuracy of 93% [3]. As shown in Section 3, our manual analysis also proves its high accuracy and precision.

External validity. This study focuses on the license inconsistency issues in Debian. The results and analysis may not be generated to other software systems. And we plan to study other software systems in the future.

5 Related Work

Many studies in software engineering have been done on software license. Some approaches for software license identification have been proposed [3,6,8]. Using these approaches, some researches analyzed software licenses in open source projects and revealed some license issues. Di Penta et al. 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 [2]. German et al. proposed a method to understand licensing compatibility issues in software packages [4]. They mainly focused on the compatibility between license declared in packages

and those in source files. In another research, 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 [5]. Alspaugh et al. proposed an approach for calculating conflicts between licenses in terms of their conditions [1]. Vendome et al. 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 [9]. In their following research [10], they investigated the reasons on when and why developers adopt and change licenses during evolution of FOSS Java projects on GitHub by conducting a survey with the relevant developers. They concluded that developers consider licensing as an important task in software development.

In our previous research [11], we proposed a method to detect license inconsistencies in large-scale FOSS projects. We then applied this method to Debian 7.5 and examined the results manually and discovered various reasons that caused these license inconsistencies, among which some were legally suspicious and deserved further investigation. As far as we know, no research has been done to investigate the evolution of license inconsistencies.

6 Conclusions

In this research we have applied our license inconsistency detection method to both Debian 7.5 and Debian 8.2. By comparing the results of these two versions of Debian, we identified three evolution patterns of license inconsistencies. With a manual analysis of the license inconsistency cases we discovered various reasons that caused the evolution. Although license inconsistencies are detected in both versions of Debian, from our manual analysis we concluded that these reported license inconsistencies are caused because the upstream project updated the license while the downstream project has not been synchronized yet. These findings suggest that in our target ecosystem, Debian, license inconsistencies are caused by distribution latency and they will disappear when the downstream projects get synchronized.

For future work, we would apply this approach to other software ecosystems and see whether there are different patterns of license inconsistency evolution and whether the license inconsistency issues are properly handled.

Acknowledgments

This work is supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (S) “Collecting, Analyzing, and Evaluating Software Assets for Effective Reuse”(No.25220003) and Osaka University Program for Promoting International Joint Research, “Software License Evolution Analysis”.

References

1. Alspaugh, T., Asuncion, H., Scacchi, W.: Intellectual property rights requirements for heterogeneously-licensed systems. In: Proceedings of the 17th International Requirements Engineering Conference (RE2009). pp. 24–33 (2009)
2. Di Penta, M., German, D.M., Guéhéneuc, Y.G., Antoniol, G.: An exploratory study of the evolution of software licensing. In: Proceedings of the 32nd International Conference on Software Engineering (ICSE2010). pp. 145–154 (2010)
3. German, D.M., Manabe, Y., Inoue, K.: 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)
4. German, D., Di Penta, M., Davies, J.: 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)
5. German, D., Di Penta, M., Gueheneuc, Y.G., Antoniol, G.: 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)
6. Gobeille, R.: The FOSSology project. In: Proceedings of the 5th Working Conference on Mining Software Repositories (MSR2008). pp. 47–50 (2008)
7. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering 28(7), 654–670 (2002)
8. Tuunanen, T., Koskinen, J., Krkkinen, T.: Automated software license analysis. Automated Software Engineering 16(3-4), 455–490 (2009)
9. Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., Germán, D.M., Poshyvanyk, D.: 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)
10. Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., German, D.M., Poshyvanyk, D.: When and why developers adopt and change software licenses. In: Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on. pp. 31–40. IEEE (2015)
11. Wu, Y., Manabe, Y., Kanda, T., German, D.M., Inoue, K.: Analysis of license inconsistency in large collections of open source projects. Empirical Software Engineering pp. 1–29 (2016)