Empir Software Eng (2017) 22:1194–1222
DOI 10.1007/s10664-016-9487-8

# Analysis of license inconsistency in large collections of open source projects

**Yuhao Wu[1] · Yuki Manabe[2] · Tetsuya Kanda[3] · Daniel M. German[4] · Katsuro Inoue[1]**

Published online: 3 December 2016


**Abstract** Free and open source software (FOSS) plays an important role in source code reuse practice. They usually come with one or more software licenses written in the header part of source files, stating the requirements and conditions which should be followed when been reused. Removing or modifying the license statement by re-distributors will result in the inconsistency of license with its ancestor, and may potentially cause license infringement. In this paper, we describe and categorize different types of license inconsistencies and propose a method to detect them. Then we applied this method to Debian 7.5 and a collection of 10,514 Java projects on GitHub and present the license inconsistency cases found in these systems. With a manual analysis, we summarized various reasons behind these license inconsistency cases, some of which imply potential license infringement and require attention from the developers. This analysis also exposes the difficulty to discover

Communicated by: Romain Robbes, Martin Pinzger and Yasutaka Kamei

✉ Yuhao Wu
wuyuhao@ist.osaka-u.ac.jp

Yuki Manabe
y-manabe@cs.kumamoto-u.ac.jp

Tetsuya Kanda
t-kanda@is.naist.jp

Daniel M. German
dmg@uvic.ca

Katsuro Inoue
inoue@ist.osaka-u.ac.jp

[1] Graduate School of Information Science and Technology, Osaka University, Osaka, Japan

[2] Faculty of Advanced Science and Technology, Kumamoto University, Kumamoto, Japan

[3] Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan

[4] Department of Computer Science, University of Victoria, Victoria, Canada

license infringements, highlighting the usefulness of finding and maintaining source code provenance.

**Keywords** Software license · Code clone · License inconsistency

## 1 Introduction

Software reuse has long been advocated as a good practice to reduce development time and increase product quality (McIlroy et al. 1968; Standish 1984; Boehm 1987). The popularity of Free and Open Source Software (FOSS) has made software reuse a common practice. FOSS software can be defined as software that is licensed under a free or open source license. In a nutshell, a free and open source license allows the software to be freely used (as in freedom), modified, and redistributed (in modified or unmodified form) by anyone, as long as the conditions of its license are satisfied. The Open Source Initiative (OSI) has defined a set of characteristics that an open source license should have, and published a list of approved Open Source licenses[1]. The Free Software Foundation[2] defines a set of similar conditions that a license should satisfy in order to be considered a free software license.

Developers who reuse FOSS should pay special attention to the license under which a source file is made available, and make sure that they satisfy the conditions and limitations of its license. Otherwise they risk losing the right to reuse the software. Typically, the license of a file is located in the top part of the file. We will refer to this area of the file as the *license statement* of the file.

The license of a file can only be changed by its copyright owner. In some special cases, the license terms allow others to change the license of the file. Otherwise, if the license is changed there is the potential for copyright infringement. For example in a case of XimpleWare Corp v. Versata Software Inc. et al[3], Versata was sued for including GPL-licensed code into one of its products but removing the copyright and use notices required by GPL. This case was settled out of court in favor of XimpleWare.

For the purpose of this paper, we are interested in the situation where a copy of a file has a different license than the original file. If the new license has not been approved by the copyright owner we are confronted with a potential *license violation*. However, in many cases it is not clear whether the change in license has been approved by the copyright owner. For example, the copyright owner might have approved, via direct communication, a change in license. Under this scenario, the copy has a different license than the original, but it is not a license violation. For this reason, when a copy of a file has a different license than the original, we say that there is a *license inconsistency* between the licenses of the two files. Some license inconsistency cases might turn out to be license violations.

Anybody who wants to reuse FOSS software should be concerned that the software being reused is properly licensed. If the reused software contains files that have been copied from other sources, and these files have license inconsistencies, then it is important to resolve these inconsistencies. Otherwise the reuser of these files might be involved in legal disputes with the original copyright owner.

[1] http://opensource.org

[2] http://www.fsf.org

[3] http://www.ifross.org/en/artikel/versata-saga-settled-prejudice-1

Previous study by Li et al. (2009) shows that 36 % of the developers who reused the OSS components changed the source code, but they did not point out whether these changes involve the license statement. In our study, we focus on the license statement changes and the license inconsistency introduced between the different copies of the files.

To the best of our knowledge, no research has been done to discover and study the characteristics of license inconsistency in software reuse. For example, how many types of license inconsistencies are there? Do they exist in open source projects? If so, what is the proportion of each type? What caused license inconsistency?

Based on these questions, we set our research question as follows:

- **RQ1** *How can we categorize a license inconsistency?*
- **RQ2** *Do license inconsistencies exist in open source projects?*
- **RQ3** *What is the proportion of each type of license inconsistency?*
- **RQ4** *What caused license inconsistencies? Are they legally safe?*

The contributions of this paper are:

1) We describe and categorize different types of license inconsistencies.
2) Based on existing tools for license identification and clone detection, we have developed a method to detect license inconsistencies. We perform an empirical study with this method using two sets of FOSS projects. This study reveals that license inconsistencies exist. It also proved the feasibility of our method.
3) We perform a manual analysis of some license inconsistency cases to understand the reasons behind them. We then summarized these reasons into 4 categories. Among them, two categories indicate license problems and require developers' attention.

This paper is organized as follows. Section 2 describes background on FOSS licenses and license inconsistencies. Section 3 introduces our research method. Our empirical study that uses this method is described in Section 4, followed by Section 5 with a discussion of the results. Section 6 describes threats to validity. After a description of related work in Section 7, Section 8 concludes this paper and points out the future direction.

## 2 License Inconsistencies

A software license is a permission to reproduce, modify and redistribute a software, usually granted under certain conditions. An open source license is a software license that follows Open Source Definition[4] and is approved by the Open Source Initiative. As of today, only 70 licenses have been approved as Open Source License. However Black Duck Software claims that the Black Duck Knowledge Base includes over 2200 licenses[5]. Some licenses have been grouped under the same name as different versions. For example, the General Public License (GPL) has versions 1, 2 and 3. Each version is, in legal terms, a totally independent license.

To reuse OSS source code files, developers must identify the license under which the files are made available, understand their terms, and satisfy their requirements. This is not a trivial task because some open source licenses do not usually allow easy integration with software under another license (see German and Hassan (2009) for a detailed discussion on

[4] http://opensource.org/definition

[5] http://www.blackducksoftware.com/products/knowledgebase

this issue). For example, software under the Apache Public License version 2 (APL-2.0)[6] can be reused and integrated into software licensed under the GPL-3.0. On the other hand, software under the GPL-2.0 cannot be combined with software under the GPL-3.0 (however software under the GPL-2.0+, that is version 2 or any later version of the GPL, can be). Therefore, developers must know the licenses of files they reuse in order to avoid license violations.

It is also known that frequently, the source code files in an application are under different licenses (Manabe et al. 2010; Manabe et al. 2014). In addition, copies of the same file might have different licenses because the copyright owner has licensed the file accordingly. For example, the copyright owner has decided to change the license from one version of the software to the other (even if the software did not have any changes).

Confusion can arise when a developer wishing to reuse a given file finds that two or more copies of it have different licenses. Let us assume that a developer wants to reuse two copies of the same file (not necessarily identical, due to their own evolution). The first copy, copy A, has license $L_A$, and copy B has license $L_B$. If the files both came directly from the copyright owner, then it can be assumed that both files have valid licenses; but if the files came from third parties, one has to question if such parties have modified the licenses without the approval of the copyright owner (resulting in a potential license violation).

Usually, the license of an open source file is indicated in its license statement, found in the first comments of each source file. Here is an example of a license statement taken from getopt.c file in GNU library, which states that the file is under the GPL-3.0+:

```
  /* Getopt for GNU.
   * NOTE: getopt is part of the C library, so if you don't
   * know what ``Keep this file name-space'' clean means, talk
   * to drepper@gnu.org before changing it!
   * Copyright (C) 1987-1996, 1998-2004, 2006, 2008-2012 Free
   * Software Foundation, Inc.
   * This file is part of the GNU C Library.
   *
   * 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.
   *
   * This program is distributed in the hope that it will be
   *        useful, but WITHOUT ANY WARRANTY; without even the
implied
   * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
   * PURPOSE. See the GNU General Public License for more
   * details.
   *
   * You should have received a copy of the GNU General Public
   * License along with this program.
```

[6]In this paper we will use the abbreviations of FOSS licenses of the Software Package Data Exchange (SPDX), found at http://spdx.org/licenses/.

```
 * If not, see <http://www.gnu.org/licenses/>.
 *
```

Generally, the license statement of a source file can only be modified by its copyright owner. Reusers shall never modify the license statement unless it is under the permission of the copyright owner or allowed by the terms of the license.[7] Otherwise, the reusers may incur a license violation.

In order to identify potential license violations, the first step is to identify license inconsistencies between files of different projects. In the following subsections, we introduce our definition of license inconsistency and give an example of a license inconsistency we have found in Debian 7.5. Finally we categorize them based on our analysis of our two target datasets.

### 2.1 Definition

For the purpose of this research, a *license inconsistency* refers to the situation when two source files that have evolved from the same original code have license statements which include different licenses.

### 2.2 Example

In the Debian 7.5 Linux distribution, two packages, `dpkg` and `anubis`, contain a file named obstack.c. Except for the license statement, these two files are identical. For this reason we assume that these two files share the same provenance.

From package `dpkg`, the license of this file is GPL-2.0+:

```
  [...]
  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.
  [...]
```

While from package `anubis` the license is GPL-3.0+:

```
  [...]
  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.
  [...]
```

As we can see, the licenses of the two files are different: GPL-2.0+ and GPL-3.0+. The first file can be combined with software under the GPL-2.0, but the second cannot (the GPL-2.0 is incompatible with the GPL-3.0). Based on our definition, this is a case of license inconsistency. Without tracing the history of each of these files, it is not possible to determine if both licenses are valid (i.e. if the copyright owner of made the file available

[7] Some licenses, such as the Mozilla tri-license (which allowed the reuse of the file under either the MPL-1.0, the GPL-2.0+ or the LGPL-2.1) allow the user to remove one or two licenses. Similarly, files are frequently licensed with the ability to use newer versions of the license (corresponding to the + sign in the SPDX abbreviations of license names, such as GPL-2.0+).

under both licenses). The following are three of many potential scenarios that lead to this inconsistency:

1) The first file is the original one and was copied to the second project, where the license was changed from GPL-2.0+ to GPL-3.0+. In this case, because the original license allows to use newer versions of the license, the change can be done by anybody, and it is not a potential license violation.
2) The second file is the original one and was copied to the first project. The license version was changed from GPL-3.0+ to GPL-2.0+ in the first project. This could be a potential violation if the change was made without the approval of the copyright owner of the file.
3) Both of the files are copied from the same third-party project (who created the file). Each project made the copy at different times, one before, and one after the license of the file was changed by the original project copyright owner. In this case, there is no potential license violation.

To determine which one is the actual reason of the inconsistency, we need to examine the repository history of these two projects and try to determine the true origin and if possible, identify the rational for this modification of license. This topic will be discussed in Section 4.1.1.

### 2.3 Categorization

Based on the analysis of our two datasets, we observed 5 types of license evolution. They are either executed by the original author or reuser:

1) **License Addition**: The source file was without a license, and a license is added in a later time.
2) **License Removal**: The source file was under a certain license, and the license is removed in a later time.
3) **License Upgrade**: The source file was under a certain version of the GPL license—a license that allows an upgrade (such as the GPL-2.0+ and GPL-3.0+)—and it is upgraded to a newer version of the license.
4) **License Downgrade**: The source file was under a certain version of a license, and it is downgraded to an older version of the same license.
5) **License Change**: The source file was under a certain license, and it is changed to another license (except for License Upgrade and License Downgrade).

Note that, in the context of this paper in the case of license upgrade and downgrade, we only consider the GPL license family. This is because currently only the GPL licenses have a *"or later"* option (e.g. GPL-2.0+, LGPL-2.1+) which allows the reuser to choose a later version of GPL when reusing the software (i.e. to *upgrade* to a newer version). Although some other licenses, such as the Apache license, may have different versions, reusers are not allowed to choose an arbitrary version of the license. Thus it is reasonable to treat various versions of these licenses as completely different licenses. For such reason we treat the license evolution between different versions of licenses other than GPL as *license change* in this paper.

License inconsistencies are naturally caused by changes in the license of the files. We use the following types to denote different types of license inconsistencies between two files:

*LAR* License Addition or Removal. One of the two files contains a license while the other file contains no license. This type of license inconsistency is usually caused by either a license addition or a license removal. We consider both addition or removal in this inconsistency because until the provenance analysis is done, we do not know if the license was added or removed by the third party.

*LUD* License Upgrade or Downgrade. One of the two files contains a certain version of a license while the other file contains a different version of the same license. This type of license inconsistency is caused by either upgrading or downgrading the license of the file.

*LC* License Change. Two files contain different licenses (excluding *LUD* cases). This type of license inconsistency is usually caused because the license of the file was changed.

## 3 Method to Detect License Inconsistencies

In our previous work (Wu et al. 2015), we have proposed a method that can efficiently detect license inconsistencies. However, a major issue with that method is that it only considers license inconsistencies among files that have the same file name (in order to achieve a fast performance). Thus if files are renamed during the process of copy-and-own reuse, a license inconsistency will not be detected. To solve this problem and make our result cover more license inconsistency cases, we propose a new method in this paper. A detailed comparison of these two methods will be discussed in Section 5.1.

In our new approach, we focus on detecting license inconsistencies among file clones. In the scenario of source code reuse where source files are imported from an upstream project, the contents of reused source files remain almost the same, sometimes with small changes (such as modifying comments, renaming identifiers etc.) (Sasaki et al. 2010).

To decide whether source files are copies of each other—or in other words whether they share the same provenance—we compare their *normalized token sequences* (Roy et al. 2009). Normalized token sequences are generated from the source file by removing the comments, redundant white spaces, new lines, carriage returns and then converting identifiers to normalized tokens. If two files have the same normalized token sequences, then it is likely that they are copies of each other and we call them *file clones*, which are actually Type-2 code clones (Roy et al. 2009; Sasaki et al. 2010). We use `CCFinder` (Kamiya et al. 2002), a code clone detection tool, to analyze and determine if files are file clones. `CCFinder` will generate a pre-process file which contains the normalized token sequences of the source file. For those file clones with the same normalized token sequences, we assume that they come from the same origin, and then gather them into the same *file group*. Files in the same file group might have different file names but similar program statements, possibly with different comments including license statement.

Once that we group these similar files, we identify the license of the files in each group. In our approach we used `Ninka` to detect the license of source files, since `Ninka` is reported to have the highest precision of all the license detection tools including `FOSSology`, `ohcount` and `OSLC` in the research by German et al. (2010b). `Ninka` is a sentence-based license detection tool which can identify 110 different licenses with 93 % accuracy, and it can handle more than 600 files per minute. There are two special results from `Ninka`: one is *UNKNOWN*, which represents that `Ninka` has found a license but does not recognize it. The other one is *None*, which states that the source file has no license.

We then compare the licenses of each file in the license list of each group. If all the files have no license, or all of them have the same license, then there is no license inconsistency. Otherwise, the group is likely to contain one or more types of license inconsistencies. And then, based on the relation between licenses, our approach identifies the type of license inconsistency. Note that a group may have multiple types of license inconsistencies. For example, if a group consists of a file under GPL-2.0+, a file under GPL-3.0+ and another file under Apache-2.0, then the group has two types of license inconsistencies: `LUD` between GPL-2.0+ and GPL-3.0+, `LC` between GPL-2.0+/GPL-3.0+ and Apache-2.0. For such reason, we calculate *License Inconsistency Metrics* for each of these groups, from which we can measure what type of license inconsistency and how many of each type exist in the groups.

### 3.1 License Inconsistency Metrics

The following 5 metrics are introduced to help measure the license inconsistencies for a file group:

#File: Number of files in this group.

#Lic: Number of different licenses in this group. If there are two or more licenses found, then it is likely that there is a license inconsistency. If no license, or only one license is found, then all the files are either without license, or they have the same license.

#Unknown: Number of files with an unknown license in this group. For our purposes we consider all the files with unknown licenses as if they have the same license (this might under-estimate the number of license inconsistencies).

#None: Number of files without any license in this group. If $\#None > 0$ and $\#Lic > 0$ then it is possible that at least one file in the group had its license added or removed (i.e. `Lar` inconsistency).

#GPL: Number of licenses in GPL family (any version of the LGPL, GPL or AGPL licenses). This metric allows us to identify `Lud` in the GPL family.

These metrics are calculated for each file group based on their license lists. The strategies shown in Table 1 enable us to decide whether a certain type of license inconsistency exists in this group.

Specifically, if we query the metrics result for each group based on the conditions of $\#None > 0$ and $\#Lic > 0$, which mean respectively that there is one or more files with no license(s), and that there is one or more files with a license, we get what we define as `LAR` (a license addition or removal); if we query for those whose $\#GPL \geq 2$, a condition which means that there are two or more different licenses in the GPL family (such as GPL-2.0+ and GPL-3.0+), we get `LUD` (a license upgrade or downgrade); and if we query for those based on $\#GPL \leq 1$ and $\#Lic \geq 2$, which mean respectively that there are more than two licenses in this group and that there is no more than one GPL license (excluding `LUD` cases), we get `LC` (a license change) where one license is changed to another one.

**Table 1** Strategies to decide whether a certain type of license inconsistency exists in a group

| Inconsistency Type | Strategy |
|---|---|
| `LAR` | $\#None > 0$ and $\#Lic > 0$ |
| `LUD` | $\#GPL \geq 2$ |
| `LC` | $\#GPL \leq 1$ and $\#Lic \geq 2$ |

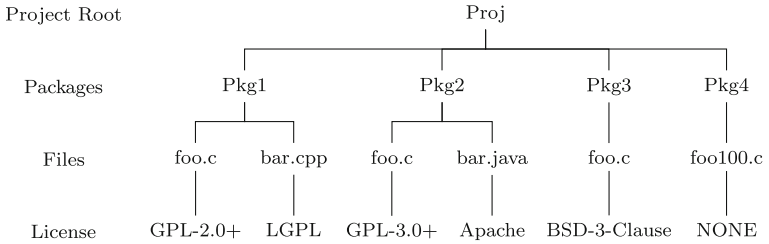



**Fig. 1** Hierarchy of a project and the license of each source file. Note that the foo.c file in `Pkg1` was imported to `Pkg2` with the license changed to GPL-3.0+; The foo.c in `Pkg3` contains totally different source code than the one in `Pkg1`, and was imported to `Pkg4` with its name changed to foo100.c and license removed

### 3.2 Method of Detecting License Inconsistencies

As a summary, our method is divided into 3 steps:

1. **Create groups of file clones:** For all the source files in the target projects, we apply `CCFinder` to extract the normalized token sequences of each file. 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. 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` is used to identify the license(s) of each file. 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.

### 3.3 Example

We illustrate our method with a project shown in Fig. 1. This project consists of 4 packages. The source code of foo.c file in `Pkg2` is exactly the same with the one in `Pkg1`, but the license statement is changed from GPL-2.0+ to GPL-3.0+; The source code of foo.c in `Pkg3` is different from the one in `Pkg1`, i.e. they happen to have the same file name. It is reused in `Pkg4` with its name changed to foo100.c and license statement removed.

1. **Create groups of file clones:** In this step, we use `CCFinder` to generate token files for each source file. Since the foo.c file from `Pkg1` and `Pkg2` have the same source

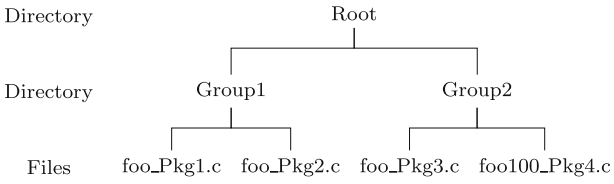



**Fig. 2** Hierarchy of the grouped files

**Table 2** License list of the selected files from the example project

| File name | GroupID | Package name | License |
|---|---|---|---|
| foo.c | 1 | Pkg1 | GPL-2.0+ |
| foo.c | 1 | Pkg2 | GPL-3.0+ |
| foo.c | 2 | Pkg3 | BSD-3-Clause |
| foo100.c | 2 | Pkg4 | NONE |

code (except for their code comments which include license statement), `CCFinder` treats them the same, and generate the same token file. This also applies to foo.c file from `Pkg3` and foo100.c from `Pkg4`. Thus we can compare the hash value of the token files and group them into two groups, as shown in Fig. 2.

2. **Identify licenses for files in each group:** For each file in the group, we use `Ninka` to detect their licenses and make a list of the file name, group index and the licenses, as shown in Table 2. *File name* is the name of the source file. *GroupID* indicates the index we use to identify file groups.
3. **Report groups that contain license inconsistencies and calculate inconsistency metrics:** We examine the licenses of each group and found that both of these groups contain license inconsistencies. Thus we report both of these groups and compute the inconsistency metrics for each of them, as shown in Table 3.

According to our rule, $\#GPL \geq 2$ in Group 1 indicates a case of *LUD* in this group, while $\#None > 0$ and $\#Lic > 0$ in Group 2 indicates a case of *LAR* in this group. This conclusion is consistent to the scenario in our example project, since the two foo.c files in `Pkg1` and `Pkg2` contain GPL-2.0+ and GPL-3.0+ respectively which is *LUD*, and the file foo.c in `Pkg3` and foo100.c in `Pkg4` contain BSD-3-Clause and no license respectively which is *LAR*.

## 4 Empirical Study

We have selected two target datasets for analysis: Debian 7.5 Linux distribution[8] and 10,514 Java projects randomly downloaded from GitHub[9]. We then conducted our method on both datasets respectively. Since it is hardly feasible to determine how many and what types of license inconsistencies are there in the target projects, it is difficult to get an oracle data set and to perform a quantitative evaluation of our method, specially regarding its recall, which we will talked more about in Section 5.4. However, a qualitative evaluation of this method is discussed in Section 5.

The following subsections will present the results obtained from the two datasets, respectively.

### 4.1 Empirical Study on Debian 7.5

We conducted our study using a large open source Linux distribution, Debian 7.5. The source code was downloaded from its official site and its main characteristics are shown in

[8]https://www.debian.org/

[9]https://github.com/

**Table 3** List of the license inconsistency metrics for each file group in the example project

| GroupID | #File | #Lic | #None | #Unknown | #GPL |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 0 | 0 | 2 |
| 2 | 2 | 1 | 1 | 0 | 0 |

Table 4. Only .cpp, .c and .java files are used, since they account for the majority of source code in the Debian distributions and are the file formats supported by `CCFinder`.

### *4.1.1 Results*

In the first step, we grouped the files under each set by their normalized token sequences and resulted in 125,092 groups in total. The number of files within one group ranges from 2 to 160, and the average number of files per group is 2.8 with a median value of 2. The breakdown of each file type is shown in Table 5.

Completing the following two steps, 6,763 groups were reported to have at least one type of license inconsistency, which is 5.4 % of the 125,092 groups in total. For the sake of space, we show only three of them in Table 6, representing each of the three types of license inconsistencies, which will be discussed in the rest of this section.

Then we calculate the number of each type of license inconsistency and their proportion. The result is shown in Table 7. From this table, we can see that from the total of 6,763 groups that contain one or more license inconsistency cases, 67.5 % of them contain *LC*, followed by *Lud* and then *Lar*. Further study is needed to investigate the legality of these modifications.

In the following paragraphs, we show examples for each type of license inconsistency.

– ***LAR*:**

Examining the getopt.c in the second line from the inconsistency result list in Table 6, we get the license list of that group in Table 8. The remaining files that contain the same licenses are omitted from this list.

We can see that the license of the getopt.c file from the `iceweasel` package has an MPL-2.0 license while the one from package `icedove` has no license (marked as NONE). The contents of each file is as follows.

getopt.c from `icedove` package:

```
#include <stdio.h>
#include <string.h>
[...]
int main(int argc, char **argv)
{
```

**Table 4** Main characteristics of Debian 7.5

| Characteristics | Number |
|---|---|
| Source Packages | 17,160 |
| Total files | 6,136,637 |
| .c files | 472,861 |
| .cpp files | 224,267 |
| .java files | 365,213 |

**Table 5** Breakdown of number of groups and files for each type in analyzing Debian 7.5

| File type | Group count | File count | #Files(mean) | #Files(median) |
|---|---|---|---|---|
| .c | 68,568 | 207,620 | 3.0 | 2 |
| .cpp | 16,202 | 38,617 | 2.4 | 2 |
| .java | 40,322 | 108,868 | 2.7 | 2 |
| Total | 125,092 | 355,105 | 2.8 | 2 |

```
        PLOptState *opt;
        PLOptStatus ostat;
        [...]
        return 0;
    }
```

getopt.c from iceweasel package:

```
  /* This Source Code Form is subject to the terms of the
  * Mozilla Public License, v. 2.0. If a copy of the MPL
  * was not distributed with this file, You can obtain
one
  * at http://mozilla.org/MPL/2.0/.
  */
  #include <stdio.h>
  #include <string.h>
  [...]
  int main(int argc, char **argv)
  {
      PLOptState *opt;
      PLOptStatus ostat;
      [...]
      return 0;
  }
```

As we can see in the file from icedove package, there is no license statement at all, while the file getopt.c from iceweasel package contains a MPL-2.0 license. Meanwhile, the other parts of these two files are exactly the same, hence we consider

**Table 6** Partial list of the license inconsistency metrics for each file group in detecting Debian 7.5

| File name* | GroupID | #File | #Lic | #None | #Unknown | #GPL |
|---|---|---|---|---|---|---|
| obstack.c | 6645 | 19 | 2 | 0 | 0 | 2 |
| getopt.c** | 46474 | 6 | 2 | 3 | 0 | 0 |
| getopt.c** | 52662 | 9 | 2 | 1 | 7 | 1 |
| ... | ... | ... | ... | ... | ... | ... |

* Each group may contain files with different file names. In this case we choose the majority file name to represent that group.
** These two groups both contain files named getopt.c, but the source code between these two groups are totally different

**Table 7** Number of different types of license inconsistencies and their proportion in Debian 7.5. Note that one group may contain more than one inconsistency types, so that the total percentage can exceed 100 %

| Inconsistency type | Frequency | Perc. |
|---|---|---|
| *LC* | 4,562 | 67.5 % |
| *LUD* | 2,137 | 31.6 % |
| *Lar* | 883 | 13.1% |

it safe to assume that the origin of both files is the same. There are several possible explanations to this case of license inconsistency:

1. The file from `icedove` package is the original one, and the developers of `iceweasel` project reused the file and added a license to it.
2. The file from `iceweasel` package is the original, and developers of `icedove` project reused this file and removed the license statement.
3. Both of the files in these two projects reused different versions of this file from another project (where the license was added or removed).

One way to try to discover which one is the true explanation is to look at the history of the files in their corresponding version control repositories. By tracing the revision history of both files, we found that the actual history reflects the third possible explanation: the files in these two projects were imported from a third project named `nspr`, where the getopt.c file was created without a license in version 4.7.1, and, for version 4.9.1 the license was changed to the MPL-2.0. It seems that `icedove` reused this file before the license statement was added, while `iceweasel` imported the version after the license was added, thus caused the inconsistency of license.

– ***LUD*:**

To exemplify this type of license inconsistency, we will use obstack.c, which is in the first line in Table 6. Table 9 shows two packages that reuse this file. As we can see from this table, the first file is licensed under GPL-2.0+ while the second one is under GPL-3.0+.

The license statements of the files from `dpkg` and `anubis` package were listed in Section 2.2. Both of these files contain more than 400 lines of code, and they are exactly the same except for their license statements. Tracing the file history in both projects we found that this file was originally created in `gnulib`. The license of this file was upgraded in `gnulib` from GPL-2.0+ to GPL-3.0+. By examining the commit log of `dpkg`, we found that the developers of `dpkg` intentionally reused the older version of the file from `gnulib` project (they wanted the file to be licensed GPL-2.0+, not GPL-3.0+), which caused the license inconsistency.

– ***LC*:**

We demonstrate this type of license inconsistency using getopt.c in the third line from the Table 6.

As shown in Table 10, getopt.c from `snort` package contains GPL-2.0 while the license of the one from `sofia-sip` could not be recognized.

**Table 8** Example of *LAR* inconsistency, in getopt.c

| Package name | License |
|---|---|
| icedove | NONE |
| iceweasel | MPL-2.0 |

**Table 9** License list of group 6645 of obstack.c where *LUD* exists

| Package name | License |
|---|---|
| dpkg | GPL-2.0+ |
| anubis | GPL-3.0+ |

The contents of these files are as follows.

getopt.c file from `snort` package:

```
[...]
** it under the terms of the GNU General Public License
** Version 2 as published by the Free Software
Foundation.
** You may not use, modify or
[...]
```

getopt.c file from `sofia-sip` package:

```
[...]
* COPYRIGHTS:
*This module contains code made available by IBM
*Corporation on an AS IS basis. Any one receiving the
*module is considered to be licensed under IBM
copyrights
*to use the IBM-provided source code in any way he or
she
*deems fit, including copying it, compiling it,
modifying
[...]
```

From the header we know that the second file is licensed under IBM copyrights, but this is not a standard version of IBM Public License, thus `Ninka` reported it as UNKNOWN. Since both these files contain the same program code, we may assume that someone changed the license from one to the other. We tried to find out the direction of this change, but due to lack of history it was not possible to do so. This shows that determining the true provenance of a file is difficult in general.

### *4.1.2 Manual Analysis*

To decide whether these license inconsistency cases may indicate legal problems or not, we have conducted a manual analysis on the history of a subset of the files.

We randomly chose the samples. To be precise, first we randomly selected a case of license inconsistency and investigated the reason that this case occurs, then we randomly selected the next case and repeated the process. The time needed to investigate each case

**Table 10** License list of group 52662 of getopt.c where *LC* and *LAR* exist

| Package name | License |
|---|---|
| p0f | NONE |
| snort | GPL-2.0 |
| sofia-sip | UNKNOWN (IBM) |

varies from several minutes to several hours, depending on how well the related project is documented. Due to the difficulties and the time invested, we stopped investigating cases when the reasons are saturated, that is, when same reasons of license inconsistency kept coming up as we investigate new cases. Based on this policy, we have investigated 25 cases in total. Then we tried to categorize them according to the reason that caused such inconsistencies. They are divided into three categories: safe changes (no violation is found), unsafe changes (given all information available, it appears to be a violation) and uncertain (it was not possible to determine whether it was safe or unsafe). The results are shown in Table 11, and the a detailed explanation of each category is as follows:

– **Safe Changes:** In this category, either the original author or the developers who reused the file changed the license statement, but the change they made is based on the terms described in the license thus we classify it as a safe change. They are further divided into 2 groups:

  *1) Original author modified/upgraded the license.* In this case, the author of that file modified the license statement (either by upgrading or totally changing it to another license), while the reusers still use the old version of the file (either intentionally or unintentionally).

  For example, as mentioned above we examined a file named obstack.c in our inconsistency result. This file originates from `gnulib` project, and its license is upgraded from GPL-2.0+ to GPL-3.0+ in a commit on 10/7/2007. This file was reused in the `dpkg` project but with a GPL-2.0+ license, and in the last commit on 9/25/2011 the log is as follows:

```
  libcompat: Update obstack module from gnulib.
The version taken is the one before the switch to
GPL-3.0+. With a slight code revert to not have to
include
  exitfail.c and exitfail.h.
  [...]
```

  We can see that in this case, the reuser intentionally takes an older version from the original project, which caused the inconsistency of license.

**Table 11** The count and percentage of each category for the 25 investigated license inconsistency cases

| Category | # | Perc. | Sub-category | # | Perc. |
|---|---|---|---|---|---|
| Safe changes | 14 | 56 % | Original author changed the license. | 10 | 40 % |
| | | | Reuser chose a license from a multi-license. | 4 | 16 % |
| Unsafe changes | 6 | 24 % | Reuser changed the license. | 5 | 20 % |
| | | | Reuser added one or more licenses. | 1 | 4 % |
| Uncertain cases | 5 | 20 % | Source files are too small to be considered as clones. | 2 | 8 % |
| | | | Source files cannot be found in the upstream repositories. | 1 | 4 % |
| | | | Repositories are not available. | 2 | 8 % |
| Total | 25 | | | 25 | |

In another example, there is a file named paintwidget.cpp, which originates from Qt project with BSD-3-Clause license. In another project called PySide, this same file is licensed under LGPL-2.1/GPL-3.0 dual license. Since these two projects both belong to Digia plc, which were acquired from Nokia, this shall be a legal license modification.

2) *The file was originally multi-licensed and reusers chose either one.* The author of the file licensed the file under two or more licenses, and the reusers can choose either one of them.

   There is a file named SimpleXMLParser.java which originates from iText project and was under the Mozilla MPL-1.1/LGPL-2.0+ dual license. This license allows the removal of one license. Developers in pdftk project reused this file removing the MPL-1.1 license and chose LGPL-2.0+ as its license.

– **Unsafe Changes:** Under this category, developers who reused the source file seemed to have modified the license statement which is not allowed by the original license terms. This change may lead to legal disputes, thus we say it is an unsafe change. We should clarify that we have reached this conclusion based on the historical evidence available. The consequence is that anybody who would like to reuse these files should pay special attention to these cases, and do due diligence to determine what is the appropriate licensing of the file, and if it indeed poses a legal risk.

1) *Reuser replaced the original license, and changed the copyright owner.* The file is under a certain license in the original project and developers who reused the file changed the license statement and the copyright owner.

   From our inconsistency list, we examined a file named X. (Because we do not have certainty regarding our conclusion, we have declared not to include the names of the projects and files.) According to the copyright year of X, company Y is the copyright owner, and licensed the file under BSD-3-Clause. When reused in a project named Z, developers changed the license to GPL-2.0+ and the copyright header, which is not allowed in BSD-3-Clause. This kind of changes to the license statement by the reuser may lead to license infringement, and may involve the reuser into legal disputes.

2) *Reuser added one or more licenses.* The original file is under some licenses, and the reuser added one or more licenses to it while retaining the original license.

   From the result we examined a file named DOMException.java. This author of this file is World Wide Web Consortium (W3C), and was licensed under W3C Software License. When developers reused this source file in ikvm project, they added a GPL-2.0 License to it resulting a composition of these two licenses. Meanwhile, the program code of this file was not changed at all. We consider this case as unsafe, since this type of license modification makes it unclear which part contains the original license and which part contains the new license, since they added the license without adding any source code changes to the file.

– **Uncertain Cases:** This category contains the license inconsistency cases which are difficult to determine whether they are legally safe or not due to several reasons:

1) *Source files are too small.* Some files contain the same source code, but due to their small size it is difficult to decide whether one is reused by the other or they just happen to be the same. A more detailed case is discussed in Section 5.2.

**Table 12** Main characteristics of Java projects cloned from GitHub

| Characteristics | Number |
|---|---|
| Projects | 10,514 |
| Total files | 3,374,164 |
| .c files | 15,627 |
| .cpp files | 21,176 |
| .java files | 3,337,361 |

2) *Files cannot be found in the upstream repositories.* We found many cases of license inconsistencies in the projects in Debian 7.5 that, when investigated the upstream project's repository, the file no longer existed.

   For example, our method reported a file named jim-win32.c in `jimtlc` package with BSD-2-Clause license and in `openocd` package with Apache-2.0 license. When we tried to look for this file in the repository of `openocd` project, it was not found. One explanation is that the file was removed in the project, but was not yet updated in Debian 7.5.

3) *Project repository not available.* Some project repositories could not be found due to lack of documentation, while some could not be accessed due to server error.

   One example is, when we tried to checkout the source code of `axis` project using the SVN command found on its official website[10], the command returned an error that the URL does not exist.

## 4.2 Empirical Study on Java Projects

The other data set we studied is a collection of 10,514 Java projects randomly cloned from GitHub. The snapshot was taken in Mar. 2015, and only those projects that consist of at least 100 commits are selected. Table 12 shows the characteristics of these projects. Since .java files are 98.9 % of all the files, we will focus our following analysis on them only.

### *4.2.1 Results*

In the first step, source files are grouped by their normalized token sequences. The result was 199,284 groups. The number of files within each group ranges from 2 to 1514, and the average number is 3.9 with a median value of 2, as shown in Table 13.

With the following steps being done, 13,916 groups are reported to contain one or more license inconsistencies, which is 7.0 % of the 199,284 groups in total.

Furthermore, the number and proportion of each type of license inconsistency is shown in Table 14.

### *4.2.2 Manual Analysis*

As we did in the Debian study, we examined a random sample of the inconsistent groups. We sampled 17 cases, and tried to categorize them according to the reason that caused such

[10] https://axis.apache.org/axis/cvs.html (Last access: Oct. 2nd, 2015)

**Table 13** Number of groups and files in each group in analyzing Java projects

| File type | Group count | File count | #Files(mean) | #Files(median) |
|---|---|---|---|---|
| .java | 199,284 | 769,220 | 3.9 | 2 |

inconsistencies. As described before, they are divided into three categories, the percentage of each category is shown in Table 15, and the explanation to each category is as follows:

– **Safe Changes:**

*1)* *Source files are in the same project but with different licenses.* Some projects were imported from other version control systems, such as SVN, where branching and tagging makes copies of the whole project. When the license of source files in the main branch (trunk) changes, license inconsistency occurs among these branches.

For example, there is a project named `weka` which was imported from SVN. In this project, files were originally licensed under GPL-2.0+ and then upgraded to GPL-3.0+. Developers made a series of tags in the SVN repository, leaving several copies of the whole project. Thus license inconsistencies exists between the files under the tags which were made before the license upgrade and those in the trunk.

Some other cases are, the source files are in the same project but exist under different directories with different licenses.

*2)* *Duplicated projects are not up-to-date.* Some entire GitHub projects (or subdirectories in other cases) are a copy (clone) of another project, and their license of source code is not updated while the original project changed its license.

We examined two projects: `JCrypTool`[11] and `JCT-CA`[12]. A file named ResizeHelper.java exists in both projects with the same normalized token sequences. The one in `JCT-CA` is without a license, while the one in `JCrypTool` was originally with no license but then added with a EPL-1.0. The readme file from `JCT-CA` states:

```
JCT-CA is going to be a plugin for the JCrypTool
regarding Public Key Infrastructure. Main development
is done in the master branch, others (if any) are just
for backing up older parts of the project and keeping
master clean.
```

From this notice we can see that, this project is a partial backup of the `JCrypTool` project, but its license is not up-to-date when the original copy has changed, resulting in a license inconsistency.

*3)* *Reuser added a same license to the source file.* One rare case we found is, the developers of a reused source file, which is under Apache-2.0, added another exactly same Apache-2.0 license description in the header. One explanation is that the developers are using automated tools to man age the licenses, but did not check whether the file already contains a license. Though it does not conflict with the license terms, we consider it as a bad smell.

[11] https://github.com/jcryptool/crypto

[12] https://github.com/Kalliope/minica

**Table 14** Number of different types of license inconsistencies and their proportion in Java projects

| Inconsistency type | Number | Perc. |
|---|---|---|
| *LC* | 12,653 | 90.9 % |
| *LAR* | 6,179 | 44.4 % |
| *Lud* | 1,316 | 9.5 % |

– **Unsafe Changes:**

*1) Reusers modified the license terms.* Some developers reused the code from other projects but made some modifications to the license terms. In this case, if it is not with the permission from the original author, these modifications are unsafe.

There are two files, both named F, in project M and N. These files are originally from project O, and M is a fork of the this project. The license of this file in O is MIT, while the one in N was changed to GPL-2.0+ with link exception.

– **Uncertain Cases:**

*1) Licenses are modified outside the scope of their repositories.* There are cases that, the source files in different projects are with different licenses, but their license statements have never changed since they were imported into these repositories. Another alternate explanation is that developers downloaded the software and modified the license before the first commit into the new repository, making it impossible to track the point where the license was changed.

*2) Source files are too small.* This case is same as the one in Debian data set. This issue will be discussed in Section 6.

For example, a file named ReaderInputStream.java was found in `bingo-core` project with an Apache-2.0 license and in `hibernate-orm` project with an LGPL license. However, the source code contents of these files are quite small, which merely contains two empty constructor methods. The source code part excluding the comments is shown as following:

```
[...]
import java.io.IOException;
import java.io.InputStream;
import java.io.Reader;
public class ReaderInputStream extends InputStream {
       private final Reader reader;
```

**Table 15** The count and percentage of each category for the 17 investigated license inconsistency cases in the Java projects

| Category | # | Perc. | Sub-category | # | Perc. |
|---|---|---|---|---|---|
| Safe changes | 11 | 65 % | Source files are in the same project but with different licenses. | 8 | 47 % |
| | | | Duplicated projects are not up-to-date. | 2 | 12 % |
| | | | Reuser added a same license to the source file. | 1 | 6 % |
| Unsafe changes | 1 | 6 % | Reuser modified the license terms. | 1 | 6 % |
| Uncertain cases | 5 | 29 % | Licenses are modified outside the scope of their repositories. | 1 | 6 % |
| | | | Source files are too small. | 4 | 24 % |
| Total | 17 | | | 17 | |

```
        public ReaderInputStream(Reader reader){
                this.reader = reader;
        }
        @Override
        public int read() throws IOException{
                return reader.read();
        }
}
```

It is possible that different developers wrote the same code like this from scratch, thus it is difficult to judge whether these files are copies of each other.

### 4.3 Discussion of the Results

From these results we can see that license inconsistencies are not uncommon: in Debian 7.5, out of 125,092 file groups, 6,763 (5.4 %) of them contain one or more license inconsistency cases: *LC* has the highest proportion with 67.5 %, followed by *LUD* with 31.6 %, *Lar* comes next with 13.1 %. While in Java projects, out of 199,284 file groups, 13,916 (7.0 %) of them contain one or more license inconsistency cases: *LC* has the highest proportion with 90.9 %, followed by *LUD* with 44.4 %, *Lar* comes next with 9.5 %.

The manual analysis of several cases of license inconsistencies gives us a rough understanding of how many of these cases are safe or not. From Tables 11 and 15 we can see that, both in Debian 7.5 and Java projects we selected, unsafe and uncertain cases take up 44 % and 35 % respectively. This shows that it is not uncommon that license inconsistencies might lead to potential license violation problems.

During this process of the analysis, we also found several challenges that prevent us from automatically analyzing the history of files.

Many files in an open source project are frequently imported from other projects. It is not a trivial task to find the repositories of these upstream projects. Take the Debian distribution as an example: some of the packages contain a file indicating the repository URL of that package, but some do not. For such packages, we needed to search for the official site of the upstream project and try to find its repository URL. There are packages that appear not to use version control systems. They simply provide source code tarballs for each version on their server. In this case, we have to download each tarball and track the license change manually. This makes provenance tracing more difficult.

In some cases the change of the license statement is not recorded in the revision history because the license statement is changed (we presume) before the file is added to the repository's project. In this case, we have to check other information (e.g. on the official site of the project or in the commit comment where the file was added) to find out the reason why developers changed the license. Our results are consistent with Vendome et al. (2015), who found a lack of traceability for license changes.

Also, after we found out that the files with the same normalized token sequences in different packages contain different licenses, we have to determine where the file comes from, i.e. the original project of that file, in order to decide the direction of the license change. But to the best of our knowledge, there is no good way to find the true origin of a certain file. We address this problem by using the date of the first commit of that file as a reference. When we have two copies in different repositories, we assume that the file with the oldest commit is the original, and files with newer dates are copies of it. If the commit date is not available, e.g. when not using a version control system, we have to manually

check the comments of the source file to see if it contains information about its true origin or its license. If not, then we are not able to decide which file comes first.

### 4.4 Answering RQs

Revisiting the research questions:

- **RQ1:** *How can we categorize a license inconsistency?* We categorize license inconsistencies into these 3 types: *i*) `LAR`, which is typically caused by license addition or removal; *ii*) `LUD`, which is related to license upgrade or downgrade in the GPL family; *iii*) `LC`, which is usually caused by license change in the process of license evolution.
- **RQ2:** *Do license inconsistencies exist in open source projects?* Yes, license inconsistencies exist in open source projects. As we have shown in our empirical studies of Debian 7.5 and a large collection of Java projects on GitHub, various types of license inconsistencies were detected.
- **RQ3:** *What is the proportion of each type of license inconsistency?* In the case study of Debian 7.5, out of 125,092 file groups, 5.4 % of them contain one or more license inconsistency cases. The proportion of each type is: `Lar` (13.1 %), `Lud` (31.6 %) and `LC` (67.5 %). In the case study of Java projects, out of 199,284 file groups we selected, 7.0 % of them contain one or more license inconsistency cases. The proportion of each type is: `Lar` (9.5 %), `Lud` (44.4 %) and `LC` (90.9 %).
- **RQ4:** *What caused license inconsistencies? Are they legally safe?* The reasons that caused license inconsistencies can be summarized into these groups according to our observation:

  *i*) Original author modified/upgraded the license.
  *ii*) The file was originally multi-licensed and reusers chose either one.
  *iii*) Reuser added one or more licenses.
  *iv*) Reuser appears to have replaced the original license, and changed the copyright owner.

  We consider the last two types of modification as unsafe, which would require further analysis to determine the legal risk associated with using them.

## 5 Discussion

In this section, we show the improvement we made to the research method with a comparison between these two methods, followed by a survey on developers involved in license inconsistencies.

### 5.1 Improvement of the Method

As described in Section 3, our previous method (Wu et al. 2015) omits the cases if the files are renamed during the process of copy-and-paste reuse to achieve higher performance.

In the previous method, we assume that many copy-and-paste reuse are conducted without renaming the source files. Thus we first create file sets where each set contains source files with the same file name. And then, under each file set, we then group the files by their normalized token sequences. Finally, we identify the licenses for each file in every file group and calculate the license inconsistency metrics.

**Table 16** Comparison of two methods on Debian 7.5

| Number of groups | New method | Previous method |
|---|---|---|
| Total | 6763 | 5344 |
| Intersection[i] | 5344 | 5344 |
| Relative complement[ii] | 1419 | 0 |

[i] Intersection indicates the groups both method reported.
[ii] Relative complement indicates the groups reported in one method but not the other

In this paper, however, the new method treats all the source files as a whole set, and groups them by their normalized token sequences. Thus it should obtain a more comprehensive result of license inconsistencies.

The following two subsections compare the two methods on the two data sets we used, respectively.

#### *5.1.1 Debian 7.5*

Table 16 shows the comparison of results obtained by the two methods, for Debian 7.5.

As we can see from the table, the new method covers all the groups that the previous method reported. Besides, it also reported 1419 (21.0 %) more license inconsistency groups. As a conclusion: the result from the new method is a superset of the one from the previous method, which is consistent with our expectation.

#### *5.1.2 Java Projects*

Table 17 shows the comparison of results obtained by the two methods when applied to the Java projects in GitHub.

Again we can see from this table, the new method covers all the groups that the previous method reported. However, there are merely 22 more groups reported by the new method, from which we can infer that the renaming operations are not frequently conducted in the process of copy-and-paste code reuse in these Java projects. This also proves that our previous method is able to produce a good result in detecting license inconsistencies where rename operation are not often conducted during the process of code reuse.

### 5.2 What Appears to be a Copy Might Not be a Copy

We sent emails to the 3 development teams of the projects where unsafe license modification were found, to understand why they modified the license and whether they consider it as an illegal modification and two of them replied us. One of them claimed that they wrote the source code all from scratch, and denied that this source file was copied from somewhere else. This source file was so small which contains merely two empty constructors,

**Table 17** Comparison of two methods on Java projects

| Number of groups | New method | Previous method |
|---|---|---|
| Total | 13,916 | 13,894 |
| Intersection | 13,894 | 13,894 |
| Relative complement | 22 | 0 |

thus we believe it is possible that different developers happen to create the same file. Note that, this is not a false positive case of our method, since our method is designed to detect *license inconsistency* cases in the target projects, not the *license violation* cases. However, it stresses the need to consider a minimum size threshold, in order for these small files not be considered in the analysis.

### 5.3 Changes Were Made Under the Permission of Copyright Owner

In another case, we found that Glassfish project included some copies of Apache code. However, in Glassfish project, the license of these files are changed from Apache-2.0 to CDDL and then to a combined license: CDDL, GPL-2.0 or Apache-2.0. The reply from the Glassfish team is that, they are using an automatic tool for license maintenance, and this tool mistakenly replaced the Apache-2.0 license with CDDL. This change was reported to them, and then they discussed with people at Apache and reached an agreement that these files should be updated with the combined license mentioned above. In this case, although license inconsistency exists, the change of license is under the permission of the copyright owner, thus we consider it legally safe.

### 5.4 An Attempt in Measuring the Recall

We have attempted to search on Google with keywords *"site:bugs.debian.org license"* in expectation of getting a list of bug reports in Debian project that are related to license inconsistency issues, so that we can use them as ground truth to measure the recall of our method. However, with a manual inspection on the top 10 results returned by Google, none of these bug reports are related to the license inconsistency issues discussed in this paper. For example, some bug reports are discussing the issue that the license is missing from some source files, which can hardly be utilized in this research. Therefore, it is difficult to build a ground truth for us to measure the recall of our method.

Nevertheless, although bug reports about license inconsistency are not found in the results of Google search, license inconsistency cases do exist in reality as shown in the previous sections. Thus we believe our method is still useful in discovering potential license issues related to license inconsistency problems.

## 6 Threats to Validity

In our approach, `CCFinder` was used to obtain the normalized token sequences of the source files. We then put files into the same clone group if they have the same hash value of normalized token sequences. Although `CCFinder` itself is a clone detection tool, we do not utilize the full functionality of it, thus the accuracy of `CCFinder` is not directly related to the accuracy of our method.

Meanwhile, source code files are evolving: those that come from the same provenance may differ from each other dramatically after being modified by developers, resulting in different normalized token sequences. A possible solution to this problem might be using pairwise checking method instead, e.g. detect clones based on the similarity of each pair of source files. However, in a conventional pairwise checking method, $n^2$ pairs of source files need to be processed when there are $n$ source files in total. In our approach, we only need to calculate the hash value of the normalized token sequences of each file, and the files that have the same hash value would be naturally put into the same clone group. Thus the

time complexity of our approach would be $O(n)$ instead of $O(n^2)$ in the pairwise checking method. Due to this performance reason, we chose to use hash approach instead of pairwise checking method in this paper. And since we can still get large numbers of file groups that contain license inconsistencies using this method, we believe that it is good enough for this exploratory study.

On the other hand, during our manual analysis we found file clones that contain the same normalized token sequences, but due to their small size and simplicity, it is difficult to decide whether they are copies of each other or they were written from scratch by independent developers. If the later one is the actual case, then it would be a false positive of our result. But we believe it might be good practice to report these cases, have a manual investigation on them and ask the developers directly.

One aspect that is important to highlight is that our method relies on the ability to detect copies of files. In our previous paper (Wu et al. 2015), we found copies of files by analyzing files with the same name. In this paper we compared the normalized token sequences of files. We could also do full clone detection and consider two files to be copies of each other only if they were above certain threshold. This process would have been significantly more time consuming. Ultimately, detecting license inconsistencies is a balance between performance of the detection vs. recall. If necessary, step one of our method can be replaced with other methods that provide better recall, at the expense of being slower, and potentially require more manual analysis to filter out false positives.

It is also important to highlight that the ability to detect license inconsistencies relies heavily on having a comprehensive corpus to compare against. In this study we have used two collections of source code: Debian 7.5 and Java GitHub projects. License inconsistencies in the source code can only be found if the original code is in the corpus that is being compared against.

In the process of license identification, as we used `Ninka` to identify the license of source files, its accuracy should also be considered. German et al. reported that the accuracy of `Ninka` is 93 % (German et al. 2010b). We believe this is sufficiently high, so that the license detection result is good enough to support our analysis. In addition, we regard UNKNOWN licenses as the same license within each group, different from any other licenses. If these UNKNOWN licenses in a same group are actually different from each other, we may underestimate the number of license inconsistency cases. But this concern is mitigated according to our observation to these UNKNOWN licenses: most of those in the same group actually contain the same license statement, either a license that is not approved by OSI or a user modified version of an OSI-approved license. On the other hand, if these UNKNOWN licenses are actually the same as those recognized ones (e.g. GPL-2.0, BSD-3-Clause etc.) in the same group, this could be considered as a false positive. In this case, these UNKNOWN licenses are not exactly the same as the original license, meaning that someone must have modified the license statement (making `Ninka` not able to recognize the license). We believe that it is necessary to check whether these changes are legal or not. Thus it is reasonable to treat them as license modifications, which is consistent with our assumption. To obtain more precise results, it is necessary to improve the accuracy of license identification.

## 7 Related Work

Many studies address inconsistent changes among code clones. Krinke (2007) studied on changes applied to code clones in open source software systems and showed that half of

the changes to code clone groups are inconsistent changes and these changes are not solved if they occurred in a near version. Göde and Harder (2011) studied patterns of consecutive changes to code clone in real software systems. Some approach to find inconsistent changes are proposed (Gabel et al. 2010; Higo and Kusumoto 2014). On the other hand, Bettenburg et al. (2009) showed that only 1 % $\sim$ 4 % of inconsistent changes to code clone introduce software defects. In addition, Göde and Koschke (2011) showed that most code clones do not evolve and the number of inconsistent changes is small. Our work does not address inconsistency in changes to code clones but inconsistency among licenses under which source files including code clones are distributed.

In addition, many studies in software engineering investigated software license. Some approaches for software license identification are proposed (German et al. 2010b; Gobeille 2008; Tuunanen et al. 2009). Using these approaches, some researches analyzed software licenses in open source projects and revealed some license issues. Di Penta et al. (2010) 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. (2010a) 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 German et al. (2009), 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. (2009) proposed an approach for calculating conflicts between licenses in terms of their conditions. However, our work proposed an approach to find license inconsistencies in similar files. By investigating the revision history of these files, we summarized the factors that caused these license inconsistency cases and tried to decide whether they are legally safe or not. Zhang et al. (2010) proposed an automatic method to check license compliance problems caused by ignorance or carelessness. They use Google Code Search to discover file clones with different licenses, while our method detect file clones within our target data sets. Recently Vendome et al. (2015) 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. In their following research, Vendome et al. (2015b) 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. However, license implications or compatibility are not always clear and so they can lead to changes. In addition, other external factors, such as community, purpose of usage and use of third-party libraries also influence the projects' licensing.

## 8 Conclusion and Future Work

This paper describes and categorizes different types of license inconsistencies, some of which might lead to potential license violations. We also proposed a method to identify files that might have license inconsistencies. With the proposed method, we managed to detect all these types of license inconsistencies from two data sets of open source projects: a Linux distribution Debian 7.5 and Java projects selected from GitHub. These results show the existence of license inconsistencies in open source projects and prove the feasibility of our method.

With a manual analysis on some license inconsistency cases, we discovered that there are several reasons behind them: in some cases the copyright owner changed the license statement; sometimes reusers exercised the permission that the file license gave them to remove one or more licenses from the file; in other cases, reusers added another license to the file; and finally, reusers modified the license. Among them, the last two categories are potentially unsafe and require further investigation.

Although the time needed for manual analysis on each case is relatively long, for developers who use our method, they only need to focus on the projects they are interested in (e.g. the projects they are maintaining). Thus the files they need to check are merely a small portion of the whole population. And we consider it feasible for developers to make sure their project involves no license violations using this proposed method.

In the process of our manual analysis, we came across a great difficulty to find out the reason behind each license inconsistency case. On one hand, it is difficult to find out from where a certain file in a project is imported when lacking enough information. On the other hand, it is also not a trivial task to decide which file is the original work when they are found in multiple projects. We tried to utilize creation date of the source files as the metric to decide which one is the original one. However, different ways of duplicating files have different influences on the creation date of that duplicated file. Pulling from a git repository or simply copy-and-pasting a file will override the creation date; extracting files from an archive file (such as a zip file) will not override the creation date. Thus we consider it not sufficient to decide the provenance of a file using the creation date only. These problems highlight the need for a method to find and maintain the provenance between applications.

For future work, we will apply our tool to more open source projects and examine the proportion of each type of license inconsistency. With the increased number of projects, we believe that many more license inconsistency cases will be found. And we will try to make a quantitative evaluation of this tool. Furthermore, we will try to develop a method to help us analyze the history of each file, so that we can decide the safety of these inconsistencies efficiently.

**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

Alspaugh T, Asuncion H, Scacchi W (2009) Intellectual property rights requirements for heterogeneously-licensed systems. In: Proceedings of the 17th International Requirements Engineering Conference (RE2009), pp 24–33. doi:10.1109/RE.2009.22

Bettenburg N, Shang W, Ibrahim W, Adams B, Zou Y, Hassan A (2009) An empirical study on inconsistent changes to code clones at release level. In: Proceedings of the 16th Working Conference on Reverse Engineering (WCRE2009), pp 85–94. doi:10.1109/WCRE.2009.51

Boehm BW (1987) Improving software productivity. Computer 20(9):43–57. doi:10.1109/MC.1987.1663694

Di Penta M, German DM, Guéhéneuc YG, Antoniol G (2010) An exploratory study of the evolution of software licensing. In: Proceedings of the 32nd International Conference on Software Engineering (ICSE2010), pp 145–154

Gabel M, Yang J, Yu Y, Goldszmidt M, Su Z (2010) Scalable and systematic detection of buggy inconsistencies in source code. In: Proceedings of the 25th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA2010), pp 175–190

German D, Di Penta M, Gueheneuc YG, Antoniol G (2009) 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. doi:10.1109/MSR.2009.5069483

German D, Di Penta M, Davies J (2010a) Understanding and auditing the licensing of open source software distributions. In: Proceedings of the 18th International Conference on Program Comprehension (ICPC2010), pp 84–93. doi:10.1109/ICPC.2010.48

German DM, Hassan AE (2009) License integration patterns: Addressing license mismatches in component-based development. In: Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, IEEE, pp 188–198

German DM, Manabe Y, Inoue K (2010b) 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

Gobeille R (2008) The FOSSology project. In: Proceedings of the 5th Working Conference on Mining Software Repositories (MSR2008), pp 47–50

Göde N, Harder J (2011) Oops! . . . I changed it again. In: Proceedings of the 5th International Workshop on Software Clones (IWSC2011), pp 14–20

Göde N, Koschke R (2011) Frequency and risks of changes to clones. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE2011), pp 311–320. doi:10.1145/1985793.1985836

Higo Y, Kusumoto S (2014) MPAnalyzer: A tool for finding unintended inconsistencies in program source code. In: Proceedings of the 29th International Conference on Automated Software Engineering (ASE2014), pp 843–846

Kamiya T, Kusumoto S, Inoue K (2002) CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans Softw Eng 28(7):654–670

Krinke J (2007) A study of consistent and inconsistent changes to code clones. In: Proceedings of the 14th Working Conference on Reverse Engineering (WCRE2007), pp 170–178. doi:10.1109/WCRE.2007.7

Li J, Conradi R, Bunse C, Torchiano M, Slyngstad O, Morisio M (2009) Development with off-the-shelf components: 10 facts. IEEE Softw 26(2):80–87. doi:10.1109/MS.2009.33

Manabe Y, Hayase Y, Inoue K (2010) Evolutional analysis of licenses in FOSS. In: Proceedings of the Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (IWPSE-EVOL2010), pp 83–87. doi:10.1145/1862372.1862391

Manabe Y, German D, Inoue K (2014) Analyzing the relationship between the license of packages and their files in free and open source software . In: Proceedings of the 10th International Conference on Open Source Systems (OSS2014), pp 51–60. doi:10.1007/978-3-642-55128-4_6

McIlroy MD, Buxton J, Naur P, Randell B (1968) Mass-produced software components. In: Proceedings of the 1st International Conference on Software Engineering (ICSE1968), pp 88–98

Roy CK, Cordy JR, Koschke R (2009) Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Sci Comput Program 74(7):470–495

Sasaki Y, Yamamoto T, Hayase Y, Inoue K (2010) Finding file clones in FreeBSD ports collection. In: Proceedings of the 7th Working Conference on Mining Software Repositories (MSR2010), pp 102–105

Standish TA (1984) An essay on software reuse. IEEE Trans Softw Eng SE-10(5):494–497. doi:10.1109/TSE.1984.5010272

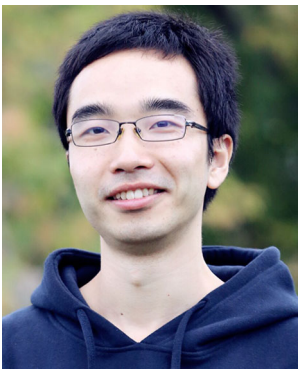
Tuunanen T, Koskinen J, Kärkkäinen T (2009) Automated software license analysis. Autom Softw Eng 16(3-4):455–490. doi:10.1007/s10515-009-0054-z

Vendome C, Linares-Vásquez M, Bavota G, Di Penta M, Germán DM, Poshyvanyk D (2015) License usage and changes: A large-scale study of java projects on github. In: The 23rd IEEE International Conference on Program Comprehension, ICPC

Vendome C, Linares-Vásquez M, Bavota G, Di Penta M, German DM, Poshyvanyk D (2015b) When and why developers adopt and change software licenses. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 31–40

Wu Y, Manabe Y, Kanda T, German DM, Inoue K (2015) 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

Zhang H, Shi B, Zhan L (2010) Automatic checking of license compliance. In: 2010 IEEE International Conference on Software maintenance (ICSM). IEEE, pp 1–3



**Yuhao Wu** received his B.E. degree (2011) of software engineering from Tongji University, and his M.E. degree (2015) of information science and technology from Osaka University. At present he is a Ph.D. student in the Graduate School of Information Science and Technology at Osaka University. His research interests include code clone detection, mining software repositories and software license.

**Yuki Manabe** received the Ph.D. in Information Science and Technology from Osaka University in 2011. He is an assistant professor at the Kumamoto University from 2013. His research interests include open source software license, open source software development and software repository mining.

**Tetsuya Kanda** is a postdoctoral fellow at Nara Institute of Science and Technology (NAIST), Japan. He received his Ph.D. in Information Science and Technology from Osaka University, Japan in 2016. His research focuses on the area of code evolution, software reuse, and source code analysis.

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

**Katsuro Inoue** is a professor of Department of Computer Science, Graduate School of Information Science and Technology, Osaka University. His current research interest includes software ecosystem modeling, software provenance analysis, and code clone detection.