A Method to Detect License Inconsistencies in Large-Scale Open Source Projects

Yuhao Wu*, Yuki Manabe†, Tetsuya Kanda*, Daniel M. German‡*, Katsuro Inoue*
* Graduate School of Information Science and Technology, Osaka University, Japan
Email: {wuyuhao, t-kanda, inoue}@ist.osaka-u.ac.jp
† Graduate school of Science and Technology, Kumamoto University, Japan
Email: y-manabe@cs.kumamoto-u.ac.jp
‡ Department of Computer Science, University of Victoria, Canada
Email: dmg@uvic.ca

***Abstract*—The reuse of free and open source software (FOSS) components is becoming more and more popular. They usually contain one or more software licenses describing the requirements and conditions which should be followed when been reused. Licenses are usually written in the header of source code files as program comments. Removing or modifying the license header by re-distributors will result in the inconsistency of license with its ancestor, and may potentially cause license infringement. But to the best of our knowledge, no research has been devoted to investigate such kind of license infringements nor license inconsistencies. In this paper, we describe and categorize different types of license inconsistencies and propose a feasible method to detect them. Then we apply this method to Debian 7.5 and present the license inconsistencies found in it. With a manual analysis, we summarized various reasons behind these license inconsistencies, some of which imply license infringement and require the attention from the developers. This analysis also exposes the difficulty to discover license infringements, highlighting the usefulness of finding and maintaining source code provenance.**

I. Introduction

As software reuse has long been advocated as a good practice to reduce development time and increase product quality [1], [2], [3], the activity of software reuse has become more prevalent. Free and open source software (FOSS) is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone, as defined by the Open Source Initiative (OSI). FOSS is always distributed under certain open source software licenses. In a nutshell, an open source license, which usually resides in the header comment of a source code file, grants the rights to redistribution of the software, and allows modifications and further distributions of the modified software.

Developers who reuse open source software should pay attention to the license header and are required to follow the conditions and limitations of the license. They shall never change the license without the permission of the copyright owner, otherwise there would be potential of license infringement. We use the term *license violation* to describe such scenarios that the license of the reused source file is changed illegally.

Previous study by Li et al. shows that 36% of the developers who reused the OSS components changed the source code [4], but they did not point out whether these changes involve the license header. In our study, we focus on the license header change during the evolution of open source software, which we refer to as *license inconsistency*.

In our research, license inconsistency refers to the situation that two source files that evolved from the same provenance but contain different licenses. It is caused by either the copyright owner of the source file or the developer who reused the source file (hereafter we refer to as *reuser*) modified the license header. The copyright owner has exclusive rights of their original work, thus their changes made to the license header of their original work shall be legal. But if those changes are made by reusers, we should pay more attention to them, since reusers can only change the license of the source file under the conditions of the license itself. Otherwise license violation may occur and involve the reusers into legal disputes.

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

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

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

The contributions of this paper are:

1) We describe and categorize different types of license inconsistency.
2) We propose a method to detect license inconsistencies in large-scale projects, which can show the existence and number of each type of license inconsistency inside the project.
3) We perform an empirical evaluation on our method to an FOSS project, which reveals the license inconsistencies

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.37

in the project and meanwhile proves feasibility of our method.

4) We perform a manual analysis on some license inconsistency cases by checking the history of the involved projects. We summarized the reasons that caused license inconsistencies, among which one is legally unsafe and needs for the developers' attention.

This paper is organized as follows. Section II describes the background knowledge of license and license inconsistency. Section III introduces our research method. An empirical study with this method is described in Section IV, followed by Section V discussing about the results. Section VI describes the threats to validity. After an introduction to the related work in Section VII, Section VIII concludes this paper and points out the future direction.

II. License Inconsistency

Software license is a permission and restriction to reproduce, modify and redistribute a software. An open source license is a software license that follows Open Source Definition[1]and is approved by Open Source Initiative. Now, 70 licenses are approved as Open Source License and BlackDuck claims that the Black Duck Knowledge Base includes data related to over 2200 licenses[2]. Some software licenses have different versions. For example, General Public License (GPL) has versions 1, 2 and 3. In addition, they allow to use "or later" which allow us to regard the version as specified version or a newer version. Some project hosting services such as SourceForge.net show license on each project.

To reuse OSS source code files, developers must understand those licenses and check whether those licenses have a conflict with the license under which the developed software is distributed. This is not a trivial task because one Open Source License is not always compatible with another. For example, GPLv3 is compatible with any version of GPL, Apachev2.0, modified BSD license (BSD3)[3]. On the other hand, GPLv3 is not compatible with Affelo GPLv3, Apachev1.0, v1.1 and original BSD license (BSD4). Therefore, license violation may occur when developers misunderstand the license of source files.

However, not all of the source code files in an application are not under the same license [5], [6]. In addition, files with the same name or the same content may have different licenses. When a developer reuses a source file, if the names and the contents of source files are similar to each other, developer may think that they are distributed under the same license. However, if these licenses are different, the developer may suffer from license violation by reusing one of them.

Usually, the license of open source software is indicated in the first comments of each source file. Here is an example of GPLv3[4] license header taken from getopt.c file in GNU library:

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

Generally, the license header of a source file can only be modified by its copyright owner. Reusers shall never modify the license header unless it is under the permission of the copyright owner or allowed by the terms of the license. Otherwise, they reusers may suffer from license violations.

To discover the potential license violations, we first find out the license inconsistencies in our research. In the following subsections, we introduce our definition of license inconsistency and give an example of license inconsistency we found in Debian 7.5. Finally we categorize them based on our analysis of the Debian project.

A. Definition

In this research, *license inconsistency* refers to the situation that two source files that evolved from the same provenance but contain different licenses.

B. Example

In the Debian 7.5 project, package `dpkg` and `anubis` both contain a file named obstack.c. Except for the license header, these two files are identical, from which we can assume that these two files share the same provenance.

From package `dpkg`, the license of this file is:

```
[...]
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:

```
[...]
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: the first one with GPLv2+, while the second one with GPLv3+. Based on our definition, this is a case of license inconsistency. Without tracing the history of each of these files, it is hard to determine which one is the origin and which one is (or both of them are) copied and modified. In this particular case, these

[1]http://opensource.org/definition
[2]http://www.blackducksoftware.com/products/knowledgebase
[3]https://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses
[4]http://www.gnu.org/licenses/gpl-3.0.html

two licenses differ only by the version number of the license. This can be explained in many different ways, some of them are:

- The first file is the original one and was copied to the second project with the license version downgraded from 2 to 3.
- The second file is the original one and was copied to the first project with the license version upgraded from 3 to 2.
- Both of the files are copied from another projects, and the license was changed to GPLv2+ and GPLv3+ respectively.

To determine which one is the real case, we need to examine the repository history of these two projects and find out the point that the change of license header happens, which will be discussed in Section IV-B.

C. Categorization

Based on the analysis to Debian 7.5, 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 release.
2) **License Removal**: The source file was under a certain license, and the license is removed in a later release.
3) **License Upgrade**: The source file was under a certain version of GPL license, and it is upgraded to a higher version of GPL license.
4) **License Downgrade**: The source file was under a certain version of GPL license, and it is downgraded to a lower version of GPL license.
5) **License Change**: The source file was under a certain license, and it is changed to another license.

Note that, in the case of license upgrade and downgrade, we only consider the GPL license. This is because currently only GPL license has an *"or later"* option which allows the reuser to choose a later version of GPL as the license for redistribution (i.e. to *upgrade* to a higher version). Although some other licenses, such as 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 our research.

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

`LAR` One of the two files contains a license while the other file contains no license. This type of license inconsistency is usually caused by License Addition or Removal in the process of license evolution.

`LUD` One of the two files contains a certain version of GPL license while the other file contains another version of GPL license. This type of license inconsistency is typically caused by License Upgrade or Downgrade in GPL license in the process of license evolution.

`LC` Two files contain different licenses. This type of license inconsistency is usually caused by License Change in the process of license evolution.

III. Method

In our approach, we focus on detecting the license inconsistencies among file clones. In the scenario of source code reuse where source files are imported from an upstream project, reused source files are kept with the same base name and their contents are almost the same, sometimes with small changes (renaming variable names etc.) [7]. But there are files that happen to have the same name but with totally different contents. For example, two individual developers may put their utility functions in a single file and both name it as util.c independently. This leads to incorrect results.

To address this problem, we decide whether they are actually from the same provenance or not by their program semantics. The rule is, if they are semantically identical, then it is likely that they are copies of each other. We use `CCFinder` [8], a code clone detection tool, to analyze and determine whether these files are semantically identical. `CCFinder` will generate a pre-process file which contains the token symbols of the source file ignoring the comments, white spaces and newlines. For those source files with the same token representation, we assume that they come from the same origin, and then gather them into the same *file group*. Files in the same file group have the same base name (but may come from different packages) and the same program statements, possibly with different comments including license header.

Now that we have gathered all these similar files, we can identify the license of each group of files and make a list of the base name and detected license for each file group. In our approach we adopted `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 this research [9]. `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. The names of the common open source licenses and their abbreviation used in this article is shown in Table I. Many of these licenses have several versions. In that case we use the suffix v<number> to identify it. If it is followed by +, that means the user can choose this version or any newer. *"or later"*: For example, GPLv2+ means *"GPL version 2 or later"*.

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 license inconsistencies. And then, based on the relation between licenses, our approach identifies the type of license inconsistency. Note

TABLE I: Names of common open source licenses and their abbreviations used in this article.

Abbrev.	Name
Apache	Apache Public License
BSD4	Original BSD, also known as BSD with 4 clauses
BSD3	BSD4 minus advertisement clause
BSD2	BSD3 minus endorsement clause
CPL	Common Public License
CDDL	Common Development and Distribution License
EPL	Eclipse Public License
GPL	General Public License
IBM	IBM Public License
LesserGPL	Lesser General Public License (successor of the Library GPL, also known as LGPL)
LibraryGPL	Library General Public License (also known as LGPL)
MIT/X11	Original license of X11 released by the MIT
MPL	Mozilla Public License

that a group may have multiple license inconsistencies. For example, if a group include a file under GPLv2, another file under GPLv3 and the other file under Apachev2, the group have two license inconsistencies: `LUD` between GPLv2 and GPLv3, `LC` between GPLv2/GPLv3 and Apachev2. For such reason, we calculate *License Inconsistency Metrics* for each of these groups, from which we can measure what type of license inconsistencies and how many of each type exist in the groups.

A. 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.

#License: 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 inconsistencies).

#None: Number of files without any license in this group. If *#License* > 0 and *#None* > 0 then it is possible that at least one file in the group had its license added or removed.

#GPL: Number of licenses in GPL family. 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 II enable us to decide whether a certain type of license inconsistency exists in this group.

Specifically, if we query the metrics result for those groups with *#None* > 0 and *#License* > 0, which means there are one or more files with no license, and also one or more files contain a license. According to our definition, this is `LAR`; If we query for those whose *#GPL* ≥ 2, which tells us that there are two or more different licenses in GPL family (such as GPLv2+ and GPLv3+), and should be `LUD`; If we query for those items with *#GPL* ≤ 1 and *#License* ≥ 2, which means there are more than two licenses appear in this group and no more than one GPL license exists (to exclude `LUD` case), it seems to be the case that one license is changed to another one, which should be a `LC`.

TABLE II: Strategies to decide whether a certain type of license inconsistency exists in a group.

Inconsistency Type	Strategy
`LAR`	*#None* > 0 and *#License* > 0
`LUD`	*#GPL* ≥ 2
`LC`	*#GPL* ≤ 1 and *#License* ≥ 2

B. Method of Detecting License Inconsistencies

As a summary, our method is divided into 4 steps:

1) **Create sets of files with same base name:** We make a list for all the .h, .c and .java files in the target project and count the occurrences of each base name. Then we create a *set* for each base name that has an occurrence larger than one. The result is sets of files with the same base name, and each set contains at least two files. Base names that contain only one file are not considered, since license inconsistency only exists between two files.
2) **Create groups of semantically identical files:** For each set of same-base-name files, we apply `CCFinder` to calculate the semantical tokens of each file. Then we create a *group* for files that have the same semantical tokens in each set. The result is groups of files that are semantically identical. This means the comments, redundant white spaces and line breaks are ignored.
3) **Identify licenses for files in each group:** For each group of semantically identical files, `Ninka` is employed to identify the license(s) of each file. The result is license lists for each file group.
4) **Report groups that contain license inconsistencies and calculate 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.

TABLE III: List of base name occurrences in the imaginary project.

Base name	Occurrence
C (.c)	
foo.c	4
C++ (.cpp)	
bar.cpp	1
Java (.java)	
bar.java	1

TABLE IV: List of the license inconsistency metrics for each file group in the imaginary project.

Base name	Group	#File	#License	#None	#Unknown	#GPL
foo.c	1	2	2	0	0	**2**
foo.c	2	2	1	**1**	0	0

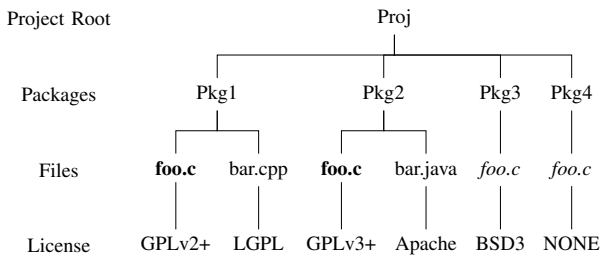

Fig. 1: Hierarchy of an imaginary project and the license of each source file. Note that the **foo.c** file in `Pkg1` was imported to `Pkg2` with the license changed to GPLv3+; The *foo.c* in `Pkg3` contains totally different source code than the one in `Pkg1`, and was imported to `Pkg4` with the license removed.

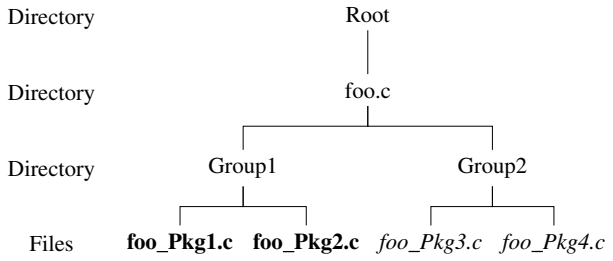

Fig. 2: Hierarchy of the grouped files.

C. Example

To make it clear, we illustrate our method with a simple imaginary project shown in Figure 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 header is changed from GPLv2+ to GPLv3+; The source code of *foo.c* in `Pkg3` is different from the one in `Pkg1`, i.e. they happen to have the same base name. It is reused in `Pkg4` with the license header removed.

1) **Create sets of files with same base name:** The calculated list of base name occurrences of the imaginary project is shown in Table III. Among them we create a set for files named foo.c, and ignore bar.cpp and bar.java since each of them only contain one file.
2) **Create groups of semantically identical files:** In this step, we use `CCFinder` to generate token files. Since the **foo.c** file from `Pkg1` and `Pkg2` have the same source code (except for their code comments which include license header), `CCFinder` treats them the same, and generate the same token file. This also applies to *foo.c* file from `Pkg3` and `Pkg4`. Thus we can compare the hash value of the token files and group them into two groups, as shown in Figure 2.
3) **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 base name, group index and the licenses, as shown in Table V.
4) **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 IV. *Base name* is the name of the source file. *Group* indicates the index we use to identify file groups.

TABLE V: License list of the selected files from the imaginary project.

Base name	Group	Package name	License
foo.c	1	Pkg1	GPLv2+
foo.c	1	Pkg2	GPLv3+
foo.c	2	Pkg3	BSD3
foo.c	2	Pkg4	NONE

According to our rule, *#GPL* > 1 in Group 1 indicates a case of *LUD* in this group, while *#None* > 0 in Group 2 indicates a case of *LAR* in this group. This conclusion is consistent to the scenario in our imaginary project, since the two **foo.c** files in `Pkg1` and `Pkg2` contain GPLv2+ and GPLv3+ respectively which is *LUD*, and the two *foo.c* files in `Pkg3` and `Pkg4` contain BSD3 and no license respectively which is *LAR*.

IV. Empirical Study

We conducted our study using a large open source Linux distribution, Debian 7.5. Since it is hardly feasible to determine how many and what types of license inconsistencies are there in the target project before our analysis, it is difficult to quantitatively evaluate our method. A qualitative evaluation of this method is discussed in Section VI and quantitative evaluation will be our future issue.

A. Setting

First, we downloaded the source code of Debian 7.5 from its official site[5]. The main characteristics of our target project Debian7.5 is shown in Table VI.

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

TABLE VI: Main characteristics of Debian 7.5.

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

TABLE VII: License list of group 10 of getopt.c where *LAR* exists.

Package name	License
icedove	NONE
iceweasel	UNKNOWN (MPLv2)

B. Results

In the first step of our method, we got 658,088 sets of base names in total. Among them, 137,618 (20.91%) sets contain more than one file, which were selected into the next step. The breakdown of the base name sets for each file type is shown in Table XI.

In step 2, we calculated the semantically identical file groups for each base name set and resulted in 74,848 such groups in total. Number of file groups per base name ranges from 1 to 160, but 91% of these sets contain only one file group, thus the average number of file groups per base name is merely 1.21.

Completing the following two steps of our method, we got the list of the final inconsistency result. From the total of 74,848 file groups, 5,359 (7.2%) of them were reported to have one or more license inconsistencies. For the sake of space, we show only part of them in Table X. Based on the strategy introduced in Table II, we calculated the number of each type of license inconsistency and their portion, as shown in Table XII.

From this table, we can see that from the total of 5,359 groups that contain one or more license inconsistencies, 98.4% of them contain *LC*, followed by *LUD* and then *LAR*. With the high distribution of *LC* in all the cases of license inconsistencies, we can see that developers are more likely to change the license of the source file to another one, which is more likely to cause license violations. For such reason, further study is urged to investigate the legality of these modifications.

TABLE VIII: License list of group 0 of obstack.c where *LUD* exists.

Package name	License
dpkg	GPLv2+
anubis	GPLv3+

TABLE IX: License list of group 15 of getopt.c where *LC* and *LAR* exist.

Package name	License
p0f	NONE
snort	GPLv2
sofia-sip	UNKNOWN (IBM)

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

1) LAR: Examining the getopt.c of group 10 in second line from the inconsistency result list in Table X, we get the license list of that group in Table VII. The rest 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 UNKNOWN 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)
{
    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 header at all, while the file getopt.c from `iceweasel` package contains a license description of MPLv2 (though `Ninka` failed to recognize it and reported as UNKNOWN). Meanwhile, the other parts of these two files are exactly the same, so we can assume that this duplicate should be caused by source file reuse. There are several possible explanations to this case of license inconsistency:

- The file from `icedove` package is the original one, and the developers of `iceweasel` project reused the file and added a license to it.
- The file from `iceweasel` package is the origin, and developers of `icedove` project reused this file and removed the license header.
- Both of the files in these two projects reused different versions of this file from another project, which caused license inconsistency.

By tracing the revision history, we found that the last one is the real case: the files in these two projects are actually imported from a third project named `nspr`, where the getopt.c file was created without a license in version 4.7.1, and was added a MPLv2 license in version 4.9.1. It seems that `icedove` project reused this file before the license header was added, while `iceweasel` project imported the version after the license was added, thus caused the inconsistency of license.

TABLE X: Partial list of the license inconsistency metrics for each file group in detecting Debian 7.5.

Base name	Group	#File	#License	#None	#Unknown	#GPL
obstack.c	0	17	2	0	0	2
getopt.c	10	5	1	2	3	0
getopt.c	15	4	2	1	2	1
...	...	...	...	...	...	...

TABLE XI: Breakdown of number of base name sets for different file types.

File Type	Total	Selected	Perc.
.c	247,520	65,572	26.49%
.cpp	154,990	24,998	16.13%
.java	255,578	47,048	18.41%
All types	658,088	13,7618	20.91%

TABLE XII: Number of different license inconsistency types and their portion in 5,359 file groups. Note that one group may contain more than one inconsistency types, so that the total percentage can exceed 100%.

Inconsistency type	Number	Perc.
LC	5,272	98.4%
LUD	2,350	43.9%
LAR	1,500	28.0%

2) LUD: Take obstack.c file of the first line from the inconsistency result in Table X as an example. The license detection result is partially shown in Table VIII. As we can see from this table, the first file is licensed under GPLv2+ while the second one is under GPLv3+.

The contents of the files from `dpkg` and `anubis` package are compared in Section II-B. Both of these files contain more than 400 lines of code, and they are exactly the same except for their license description part. Similar assumptions are possible to be made to explain this case of license inconsistency. Tracing the file history we found that this file was originally created in `gnulib`. The license of this file was upgraded from GPLv2+ to GPLv3+. By examine the commit log of `dpkg`, we found that the developers of `dpkg` intentionally reused the older version of the file from `gnulib` project, which caused the license inconsistency.

3) LC: Here we select the getopt.c files in the third line from the inconsistency result in Table X.

As shown in Table IX, getopt.c from `snort` package contains GPLv2 while the license of the one from `sofia-sip` could not be recognized.

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
[...]
```

TABLE XIII: The count and percentage of each category for the 25 investigated license inconsistency cases.

Category	Count	Perc.
Safe changes	14	56%
Unsafe changes	5	20%
Uncertain cases	6	24%
Total	25	100%

From the header we know that the second file is licensed under IBM copyrights (`Ninka` reported as UNKNOWN). Since both these files contain the same program code, we may assume that someone changed the license from one to the other but we do not know the direction yet. If these changes are not made by the original author, this may be a potential license violation.

C. Manual Analysis

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

We randomly chose the samples using trial-and-error methodology, that is, first we randomly select a case of license inconsistency and investigate whether it is legally safe or not, then we randomly select the next case and repeat the process. 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, the percentage of each category is shown in Table XIII, and the explanation to each category is as follows:

1) Safe Changes: In this category, either the original author or the developers who reused the file changed the license header, but the change they made is based on the terms described in the license thus we say it is a safe change. They are further divided into 3 groups:

Original author modified/upgraded the license: In this case, the author of that file modified the license header (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, we examined a file named obstack.c in our inconsistency result. This file originates from `gnulib` project, and its license is upgraded from GPLv2+ to GPLv3+ in a commit on 10/7/2007. This file was reused in the `dpkg` project but with a GPLv2+ 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 GPLv3+.
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.

In another example, there is a file named paintwidget.cpp, which originates from `Qt` project with BSD3 license. In another project called `PySide`, this same file is licensed under LPGLv2.1/GPLv3 dual license. Since these two projects both belong to Digia plc, which were acquired from Nokia, this shall be a legal license modification.

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 MPL/LGPL dual license. Developers in `pdftk` project reused this file removing the MPL license and chose LGPL as its license.

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 our inconsistency result we examined a file named DOMException.java. This author of this file is World Wide Web Consortium (W3C), and was licensed under W3C license. When reused in `ikvm` project, a GPLv2 License was added to it resulting a composition of these two licenses.

2) Unsafe Changes: Under this category, developers who reused the source file seemed to have modified the license header which is not allowed by the original license terms. This change may lead them to legal disputes, thus we say it is an unsafe change.

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 header and the copyright owner.

From our inconsistency list, we examined a file named SpringUtilities.java. According to the copyright year, Oracle is the copyright owner, and licensed the file under BSD3. When reused in `freemind` project, developers changed the license to GPLv2+ and the copyright header, which is not allowed in BSD3. This kind of changes to the license header by the reuser may lead to license infringement, and may involve the reuser into legal disputes.

3) 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:

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. This problem is discussed in Section VI.

Files can not be found in the repository: Although some license inconsistency cases are reported to be existing in Debian 7.5, when we investigated the project repository, the file no longer existed. One explanation is that the file was removed in the project, but was not yet updated in Debian 7.5.

Project repository not available: Some project repositories could not be found due to the lack of documentation, while some can not be accessed due to server error.

V. Discussion

From these results we can see that the license inconsistencies are not uncommon: out of 74,848 file groups, 5,359 (7.2%) of them contain one or more license inconsistencies. Among them, *LC* has the highest proportion with 98.4%, followed by *LUD* (43.9%), *LAR* comes next with 28.0%.

With a manual analysis to several cases of license inconsistencies, we discovered that some of them are possibly causing license violations, which needs further investigation. During this process of analysis, we also found several challenges that prevent us from automatically analyzing the history of files.

Packages in a large open source project are usually imported from upstream projects. It is not a trivial task to find the repositories of these upstream projects. Take Debian project 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 need have to go to the official site of the upstream project and try to get the repository URL. There are packages not even using 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.

In some cases the change of the license header is not recorded in the revision history because the license header is changed right before the file is added to the 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.

Besides, after we find out that the files with similar code contents 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 origin of a certain file. What we do in our research is to take the date of the first commit of that file as a reference. If the commit date is not available, in the situation of not using version control system, we have to manually check the comments of the source file to see if it contains information of the author. If not, then we are not able to decide which file comes first.

Revisiting the research questions:

- **RQ1:** *How can we categorize license inconsistencies?* 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 GPL family; *iii*) *LC*, which is usually caused by license change in the process of license evolution.

- **RQ2:** *Does license inconsistency exist in large open source projects?* Yes, license inconsistencies exist in large open source projects. As we studied on the FOSS project Debian 7.5, various types of license inconsistencies are detected.
- **RQ3:** *What is the proportion of each type of license inconsistency?* In the case study of Debian 7.5, out of 74,848 file groups we selected, 7.2% of them contain one or more license inconsistencies. The proportion of each type is: `LAR` (28.0%), `LUD` (43.9%) and `LC` 98.4%.
- **RQ4:** *What caused these 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 replaced the original license, and changed the copyright owner. Among them, the last type of change is unsafe.

VI. Threats to Validity

In our approach, we only consider the license inconsistencies among source files that have the *same* base name, but there might be scenarios that both the license header and the base name of the reused source file are changed. In this case, our method can not detect the license inconsistencies between the modified file and its ancestor. Since this paper is an exploratory study on license inconsistency, for simplicity and efficiency, we only focus on those cases with the same base name, which may increase the false negative. To reduce these false negatives, we can simply skip the first step of our method, which means we treat the whole project as one set, and detect file clones within all these files. This progress will surely take much more time, which we make it as our future work.

Another factor that increases false negative rate is that we use `CCFinder` to detect file clones which are exactly identical to each other regarding their semantics. However, source code files are evolving: those that come from the same provenance may differ from each other semantically after being modified by developers. But since we can still get large numbers of file groups that contain license inconsistencies using the proposed method, we believe that it is enough for this exploratory study. To mitigate this problem, we can use similarity metrics instead.

On the other hand, during our manual analysis we found files clones that are semantically identical, 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 individual developers. If the later one is the real case, then it would be a false positive of our result. But we believe it is a good practice to report these cases, have a manual investigation on them and ask the developers directly.

In the process of license identification, as we employed `Ninka` as the identification tool, the accuracy of the result from `Ninka` should also be considered. German et al. reported that the accuracy of `Ninka` is 93% [9]. 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 license inconsistencies. 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 header, 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. GPLv2, BSD3 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 one, meaning that someone must have modified the license header. 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, we need to improve `Ninka`.

VII. Related Work

Many studies address inconsistencies among code clones. Krinke [10] studied on changes applied to code clone 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 et al. [11] studied patterns of consecutive changes to code clone in real software systems. Some approach to find inconsistent changes are proposed [12], [13]. On the other hand, Bettenburg et al. [14] showed that only $1\% \sim 4\%$ of inconsistent changes to code clone introduce software defects. In addition, Göde et al. [15] 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 [9], [16], [17]. Using these approaches, some researches analyzed software licenses in open source projects and revealed some license issues. Di Penta et al. [18] 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. [19] 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 of Di Penta et al. [20], 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. [21] 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 inconsistencies and tried to decide whether they are legally safe or not. Recently Vendome et al. [22] 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.

VIII. Conclusion and Future Work

This paper describes and categorizes different types of license inconsistency. With the proposed method, we managed to detect all these types of license inconsistency from the large open source project Debian 7.5, which shows the existence of license inconsistency in open source projects and proves the feasibility of our method.

With a manual analysis on some license inconsistency cases, we discovered that there are several reasons behind license inconsistencies: Author changed the license header; Reuser chose either one from multi-licenses; Reuser added a compatible license; Reuser modified the license. Among them, the last one is potentially unsafe and needs further investigation.

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. 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 large numbers of open source projects and examine the portion of each type of license inconsistency. With the increased number of projects, we believe that much 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.

Acknowledgment

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] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell, "Mass-produced software components," in *Proceedings of the 1st International Conference on Software Engineering (ICSE1968)*, 1968, pp. 88–98.

[2] T. A. Standish, "An essay on software reuse," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 494–497, Sept 1984.

[3] B. W. Boehm, "Improving software productivity," *Computer*, vol. 20, no. 9, pp. 43–57, Sep. 1987.

[4] J. Li, R. Conradi, C. Bunse, M. Torchiano, O. Slyngstad, and M. Morisio, "Development with off-the-shelf components: 10 facts," *IEEE Software*, vol. 26, no. 2, pp. 80–87, March 2009.

[5] Y. Manabe, Y. Hayase, and K. Inoue, "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)*, 2010, pp. 83–87.

[6] Y. Manabe, D. German, and K. Inoue, "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)*, 2014, pp. 51–60.

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

[8] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, 2002.

[9] D. M. German, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *Proceedings of the 25th International Conference on Automated Software Engineering (ASE2010)*, 2010, pp. 437–446.

[10] J. Krinke, "A study of consistent and inconsistent changes to code clones," in *Proceedings of the 14th Working Conference on Reverse Engineering (WCRE2007)*, 2007, pp. 170–178.

[11] N. Göde and J. Harder, "Oops! . . . I changed it again," in *Proceedings of the 5th International Workshop on Software Clones (IWSC2011)*, 2011, pp. 14–20.

[12] M. Gabel, J. Yang, Y. Yu, M. Goldszmidt, and Z. Su, "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)*, 2010, pp. 175–190.

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

[14] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. Hassan, "An empirical study on inconsistent changes to code clones at release level," in *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE2009)*, 2009, pp. 85–94.

[15] N. Göde and R. Koschke, "Frequency and risks of changes to clones," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE2011)*, 2011, pp. 311–320.

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

[17] T. Tuunanen, J. Koskinen, and T. Krkkinen, "Automated software license analysis," *Automated Software Engineering*, vol. 16, no. 3-4, pp. 455–490, 2009.

[18] M. Di Penta, D. M. German, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the evolution of software licensing," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE2010)*, 2010, pp. 145–154.

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

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

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

[22] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. M. Germán, and D. Poshyvanyk, "License usage and changes: A large-scale study of java projects on github," in *The 23rd IEEE International Conference on Program Comprehension, ICPC 2015*, To appear.