IEICE TRANSACTIONS

on Information and Systems

VOL. E104-D NO. 2
FEBRUARY 2021

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. INF. & SYST., VOL.E104–D, NO.2 FEBRUARY 2021

PAPER

An Exploratory Study of Copyright Inconsistency in the Linux Kernel

Shi QIU[†a)], **Daniel M. GERMAN**[††b)], ***Nonmembers***, ***and*** **Katsuro INOUE**[†c)], ***Fellow***

SUMMARY Software copyright claims an exclusive right for the software copyright owner to determine whether and under what conditions others can modify, reuse, or redistribute this software. For Free and Open Source Software (FOSS), it is very important to identify the copyright owner who can control those activities with license compliance. Copyright notice is a few sentences mostly placed in the header part of a source file as a comment or in a license document in a FOSS project, and it is an important clue to establish the ownership of a FOSS project. Repositories of FOSS projects contain rich and varied information on the development including the source code contributors who are also an important clue to establish the ownership. In this paper, as a first step of understanding copyright owner, we will explore the situation of the software copyright in the Linux kernel, a typical example of FOSS, by analyzing and comparing two kinds of datasets, copyright notices in source files and source code contributors in the software repositories. The discrepancy between two kinds of analysis results is defined as copyright inconsistency. The analysis result has indicated that copyright inconsistencies are prevalent in the Linux kernel. We have also found that code reuse, affiliation change, refactoring, support function, and others' contributions potentially have impacts on the occurrence of the copyright inconsistencies in the Linux kernel. This study exposes the difficulty in managing software copyright in FOSS, highlighting the usefulness of future work to address software copyright problems.

key words: *software maintenance, mining software repositories, open source software, software copyright*

1. Introduction

Software copyright grants the copyright owner a legal right to determine under what conditions this software can be redistributed, reused, and modified with the help of software licenses. Different from proprietary software, FOSS projects are developed in a collaborative manner, receiving contributions from a large number of people, named *contributors*, from different countries or regions, or different organizations. For example, the Linux kernel is contributed by 21,074 different contributors at the end of 2019, and those contributors have the potential to claim the copyright of the contributed software [1]. In this paper, we name the individuals or organizations explicitly declared in copyright notices as the *holders*, and the individuals or organizations who actually own the copyrights as the *copyright owners*.

In general, the holders are seen as the copyright owners. However, the holders may not cover all contributors who could potentially become the copyright owners. So identifying the copyright owners of FOSS only by the holders is insufficient. Therefore, an important question in the copyright ownership problem of FOSS is: "Who are the actual copyright owners?" Identifying the copyright owner of FOSS is important for several reasons: a) only the copyright owner of FOSS is allowed to change its license or granting a commercial one to a third party, such as Oracle grants commercial licenses to MySQL* [2]; b) only the copyright owner of FOSS is allowed to start legal proceedings to enforce its license; c) several FOSS licenses (e.g. the BSD family of licenses) require that the copyright owner of FOSS projects being reused be acknowledged in the documentation and other materials of the system that reuses it.

In recent years, some developers who once contributed to large FOSS projects have enforced the open source license against the distributors who use the FOSS. These charges are based on the terms of the highly restrictive open source licenses. These distributors may face lawsuits and potential monetary penalties if they ignore or violate the terms of these restrictive licenses. For example, Patrick McHardy - a developer of the Linux kernel - has sued some companies, claiming that he shares a great part of the authorship of the Linux kernel [3], [4]. While it is easy to find his copyright notice existing in some source files, it is not easy to identify what portions of his contributions still remain in the current kernel. In other words, his declaration of the copyright notice may be inconsistent with his remained contributions. Therefore, the question arises: "Are these developers enforcing the open source license against the distributors the actual copyright owners of redistributed FOSS projects?" To address this problem, both the open source community and the industry invested a great amount of effort. For example, an open source license compliance software system and toolkit called FOSSology** have been developed to detect the copyright notice buried in the source code [5]. BlackDuck*** also provides a service of assessing the legal risks of software copyright to those who want to commercially reuse FOSS.

However, establishing copyright ownership in large

Manuscript received May 13, 2020.
Manuscript revised September 30, 2020.
Manuscript publicized November 17, 2020.

†The author is with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
††The author is with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada.
a) E-mail: qiujitsu@ist.osaka-u.ac.jp
b) E-mail: dmg@uvic.ca
c) E-mail: inoue@ist.osaka-u.ac.jp
DOI: 10.1587/transinf.2020EDP7107

*https://www.mysql.com/
**https://www.fossology.org/
***https://www.blackducksoftware.com/

FOSS projects and verifying whether the statement of a copyright notice is correct are complicated. On one hand, as we mentioned below, the contributors, who could potentially become the copyright owners as well, can not be totally covered by the holders. On another hand, since the large FOSS projects are usually co-developed by a large number of developers during a long period, the copyright notices potentially risk the poor management and can not refer to the actual copyright ownership. For example, a copyright notice can be added when a contributor contributes source code to a FOSS project, but if the contributed source code is totally deleted from this FOSS project, the added copyright notice is possibly left in the source file since the developer who did this deletion may not know which part of source code is governed by this copyright notice. In these cases, the copyright notices will denote copyright ownership incorrectly. The inconsistency between the holders in the copyright notices and the contributors of the source code (called *copyright inconsistency* or simply *inconsistency*) plays an important role in the difficulty of establishing copyright ownership in large FOSS projects.

In this paper, as a first step of understanding copyright ownership, we analyze copyright inconsistencies of the Linux kernel, which is a typical example of FOSS. To the best of our knowledge, no work has been done to go deep into copyright inconsistency. Only a few works analyzed the relationship between the contributors and the holders [6], which are insufficient to reveal some important problems about this issue such as how prevalent copyright inconsistencies are in FOSS and why copyright inconsistencies occur. Also, we want to know whether the statements of copyright notices are accurate and truly denote the contributors' contributions to the source files that contain them.

To achieve our goal, we must first identify the holders and contributors. The holders in copyright notices can be detected from the copyright notices directly. While identifying the contributors is much more difficult since tracking the "original" authors of the source code is a difficult task in software engineering as well. Here we use the *committers* who can be tracked in software repositories as an indicator of the contributors. The difference between committer and contributor is that committer is the developer who commits the source code while the contributor is the developer who actually writes the source code. Both of them could be the potential copyright owner who actually owns the copyrights. Although the committers are not always consistent with the contributors, they are the only explicit information we can observe in software repositories directly. We will explore the cases that a developer commits source code written by others in our empirical study. Using the committer can help us to conduct a quantitative analysis to study the copyright inconsistency because the holders and contributors are two most explicit pieces of information we can rely on to establish the ownership of the source file.

Table 1 summarizes all the roles introduced in this paper.

Table 1 Summary of the roles introduced in this paper.

Role	Definition
Copyright owner	Who actually owns the copyrights.
Holder	Who is explicitly declared in copyright notices.
Contributor	Who actually writes the source code.
Committer	Who commits the source code to repositories.

Based on these questions, we first define and categorize the copyright inconsistency formally. We then conduct an empirical study to study copyright inconsistency. Our research questions are set as follows:

RQ1: How prevalent are copyright inconsistencies?

RQ2: What caused the copyright inconsistencies?

The contributions of this paper are as follows:

1. We have made the first study to focus on the prevalence of copyright inconsistencies, relying a proposed analysis method to detect and study them.

2. We have also conducted an empirical study on the Linux kernel to find the reasons.

This paper is organized as follows. Section 2 first provides a brief background on copyright notices in FOSS projects, after which Sect. 3 proposes the definition of copyright inconsistency. Our empirical study on the Linux kernel is described in Sect. 4, followed by Sect. 5 with a discussion of the results. Section 6 describes threats to validity. After a description of related work in Sect. 7, Sect. 8 concludes this paper.

2. Background

In this section, we have a look at the practical situation of copyright notices and committers in the FOSS project.

2.1 Copyright Notices in FOSS Projects

A copyright notice is a sentence to declare the holders explicitly. Generally, a copyright notice begins with the word "Copyright" or a copyright sign "(C)", followed by the names of the holders. The valid year of this copyright notice is usually stated in the copyright notice as well. An example of the copyright notice in FOSS is as follows:

```
#
# Copyright (C) 2011-2013 Free Software
# Foundation, Inc.
#
# This program is free software; you can
# redistribute it and/or modify it under the
# terms of the GNU General Public License as
# published by the free Software Foundation;
# either version 2, or (at your option)
# any later version.
#
```

This copyright notice declares that the Free Software Foundation owns the copyright of this software. A FOSS license is stated with the copyright notice as well.

Usually, these holders are seen as owning the copyright. Note that in this paper, holders refer to the individu-

als or organizations explicitly declared in copyright notices. While copyright owners refer to the individuals or organizations who did own the copyrights. The holders may not always be consistent with the copyright owners.

Similar to other types of properties, copyright can be sold and transferred. Different FOSS projects have different rules on copyright transfer. Some FOSS projects require their developers to transfer the copyright of their contributions. For example, Oracle - the copyright owner of MySQL - requires all contributors to transfer the copyright of their contributions when they make the contributions [7]. This strict requirement ensures Oracle to be the only copyright owner. Therefore, "© 2019, Oracle Corporation and/or its affiliates" is the only copyright notice of MySQL, declaring that the holder, also the copyright owner, is Oracle Corporation and/or its affiliates. By having this only ownership of MySQL, Oracle is also able to offer commercial licenses.

However, most FOSS projects do not force the copyright transfer. They receive contributions from a large number of contributors, which makes establishing the ownership of FOSS more complicated. Therefore, a FOSS project may be copyrighted under from one to hundreds of copyright notices.An example is the Linux kernel where a lot of copyright notices can be observed. Meanwhile, these copyright notices are usually declared using some particular pattern. The situations of many other FOSS projects are similar to the Linux kernel.

2.2 Source Code Committers in FOSS Projects

Modern software development is usually participated by a lot of developers. The large FOSS projects are also developed in a collaborative manner, receiving commits from a large number of committers. For example, in the case of the Linux kernel, the Linux foundation - the organization dedicated to maintaining the Linux and other related FOSS projects communities - reported that over 15,600 developers from more than 1,400 companies have contributed to the Linux kernel since tracking began 11 years ago [8]. Version control systems such as git[†], usually have a feature known as "blame", to track who nominally commits certain lines of code to the repositories. To track the contribution at a more granular level of code tokens, a tool named `cregit` is developed to track the committer, time, and commit log of each token in source file [9].

In our following study, we rely on `cregit` to construct the dataset. Table 2 shows an example of the extracted information using `cregit` to a source file named generic.h in the Linux kernel. It is easy to know the list of the committers, and the proportion of the contribution of each committer. For example, Arnd Bergmann contributed 170 tokens to the source file, accounting for 80.6% of all tokens, and committed thrice, accounting for 27.3% of all commits.

[†]https://git-scm.com/

Table 2 Result of using `cregit` on generic.h.

Person	#Token	T.Prop	#Commit	C.Prop
Arnd Bergmann	170	80.6%	3	27.3%
Viresh Kumar	29	13.6%	4	36.3%
Russell King	5	2.4%	1	9.1%
Robin Holt	5	2.4%	1	9.1%
Shiraz Hashim	1	0.5%	1	9.1%
Masahiro Yamada	1	0.5%	1	9.1%
Total	211	100.0%	11	100.0%

3. Copyright Inconsistency

In this section, we make a precise definition of copyright inconsistencies and categorize them.

3.1 Definition

For the purpose of this research, copyright inconsistency refers to the inconsistency between the holders and the committer. The holders refer to the individuals or organizations explicitly declared in copyright notices according to our definition. The committers refer to the individuals who committed the source code to repositories.

3.2 Categorization

In many FOSS projects, copyright inconsistencies can be observed easily. Here we still take the source file generic.h in Linux kernel, as the target source file to observe. The header comments of this source file are as follows:

```
/*
 * spear machine family generic header file
 *
 * Copyright (C) 2009-2012
 * ST Microelectronics
 * Rajeev Kumar <rajeev-dlh.kumar@st.com>
 * Viresh Kumar <vireshk@kernel.org>
 *
 * This file is licensed under the terms of
 * the GNU General Public License version 2.
 * This program is licensed "as is" without
 * any warranty of any kind, whether express
 * or implied.
 */
```

It is easy to know that the holders are ST Microelectronics, Rajeev Kumar, and Viresh Kumar. Note that possibly Rajeev Kumar is an employee of ST Microelectronics according to the domain of his email address. We then extract the committers relying on `cregit`. The committers and the proportion of their contributions to the source files are shown in Table 2.

We can observe that inconsistencies happened in two ways.

Firstly, we can not find any hints about the ST Microelectronics and Rajeev Kumar - both seen as the holders-

in the committers. Their actual contributions to this source files are not clear. It may be a possible reason that their copyright notices are wrongly declared or just out of date.

Secondly, a lot of committers- Arnd Bergmann, Russell King, Robin Holt, Shiraz Hashim, and Masahiro Yamada - did not declare their copyright notices in the source file. This resulted in a problem that the ownership of this source file can not be established since we have no knowledge about the contributions of the committers to this source file. Especially, the copyright notice of Arnd Bergmann, who committed more than 80% source code to the source file by tokens, is not declared in the source as well. Establishing the ownership of a source file without regarding a committer who committed a large part of source code such as Arnd Bergmann is not wise.

So establishing the ownership of a source file only by the copyright notices is not sufficient. Some may argue that the committer such as Shiraz Hashim or Masahiro Yamada committed too few source codes. Therefore, the proportion of their commits can not support them to become the qualified committers and declare their ownership of this source file. We will have a discussion on this proportion in our empirical study in Sect. 4. These two types of inconsistencies between the holders and the committers can be observed in many other observed source files as well.

Based on the observation of Linux kernel, we observed two types of copyright inconsistency. One is the situation that the holder is not the committer (i.e. *holder-not-committer inconsistency*). Another one is the situation that the committer is not the holder (i.e. *committer-not-holder inconsistency*).

Note that if the organizations the committer belongs to are declared in the copyright notices, we determine there is no inconsistency in this situation. A quantitative and executable analysis of the copyright inconsistency could be conducted based on the proposed definition and categorization.

4. Empirical Study

The goal of this section is to introduce our analysis methods to answer research questions. To achieve this goal, we select the Linux kernel - the most popular and successful open-source operating system kernel - to analyze. Table 3 shows a summary of the target version of the Linux kernel.

Table 3 Summary of the target version of the Linux kernel.

Version	4.14
Date	Nov 13, 2017
#File	45,477

4.1 Research Questions

This empirical study aims at addressing the following research questions:

RQ1: How prevalent are copyright inconsistencies? This research question investigates the prevalence of copyright inconsistencies in the Linux kernel. Copyright inconsistencies will be detected using the proposed method, followed by quantitative analysis.

RQ2: What caused the copyright inconsistencies? This research question aims at finding the reasons causing the copyright inconsistencies in the Linux kernel. The results reveal the reasons by a qualitative analysis manually tracking the historical commit logs of the source files.

4.2 Dataset Construction

To achieve our goal, we first construct the datasets for our empirical study. The dataset construction is consists of three steps - sampling, building the committer dataset, and building the holder dataset.

4.2.1 Sampling

We first download the source code of the Linux kernel from Github[†]. Notice that we only collect the source files whose life cycle can be entirely traced in Github. Some source files - known as pre-git files - have been created and evolved before the source code was uploaded to Github. Those pre-git files are not our target source files. This step makes copyright notices trackable, which helps us in answering the research questions. We end up with 38,932 source files after removing pre-git files from the total downloaded source files. We aim to construct a sample dataset by randomly selecting source files from these 38,932 source files. We use a statistical method to determine the sample size needed in order to get results that reflect the target population as precisely as needed. The required sample size was calculated so that our conclusions would generalize to all 45,477 source files with a confidence level of 95% and a confidence interval of 5[††]. The calculation of statistically significant sample sizes based on population size, confidence interval, and confidence level is well established. The calculated required sample size is 381. At last, we randomly select 500 source files from 38,932 non pre-git files to construct the sample dataset.

Among these 500 source files, a part of source files' histories can not be tracked because of the mechanism of git. Git keeps track of changes to files in the working directory of a repository by their names. When a file is moved or renamed, git sees it as a creation of a new file while the original file is deleted. Since our analysis required the traceability of the entire history of a source file, we only select the source files which are not moved or renamed before. By removing source files once moved or renamed, we end up with a sample dataset consist of 414 source files. This number is still larger than 381 - the minimum required number of statistically significant sample size.

[†]https://github.com/torvalds/linux
[††]https://www.surveysystem.com/sscalc.htm

4.2.2 Building the Committer Dataset

In this step, we build the committer dataset. For each source file in the sample dataset, we extract the full name, number of tokens they contributed, and the proportion of this contribution for each committer using `cregit`. Note that a committer may have multiple different accounts in GitHub. We rely on `cregit` to solve this problem. `cregit` extracts the committer's full name and use it as the unified identifier to merge the multiple different accounts owned by the same committer. We then extract the e-mail address of each committer by tracking the historical commit logs of the source file.

Lastly, we identify the organizations the committers belong to by checking the domain of their e-mail addresses. We use a semi-automatic method to achieve this goal. A domain dictionary is built to map the domain of the e-mail addresses to the organizations. When we try to identify the organization of a committer, we first check if the domain of his or her e-mail address is in the domain dictionary or not. If so, we determine the corresponding organization as the organization the committer belongs to. Otherwise, we manually check this domain and identify the organization for this committer. The pair of the domain and the organization is added into the domain dictionary at the same time. Note that here an organization is indicated by a uniform identifier.

In this way, we build the committer dataset. All source files are contained in this dataset. For each of them, we list his or her full name, the number of tokens he or she contributed, the proportion of the contribution, e-mail address, and the organization he or she belongs to. During this process, we successfully build a domain dictionary as well.

4.2.3 Building the Holder Dataset

In this step, we build the holder dataset for each source file in the sample dataset. Note that we have built the committer dataset containing the full names of all committers and the uniform identifiers to indicate the organizations they belong to. We first use FOSSology [5] to detect the copyright notices, after which we manually check all detected copyright notices to remove the wrongly detected ones.

Then next, we build an organization dictionary shown in Table 4. The index is the uniform identifiers of the organizations. Each of the uniform identifiers refer to a list of possible organization names found in the analysis. The organization names are the different ways one organization might be referred to in copyright notices. In the beginning, the list only contains one name that we find in building the committer dataset.

We then match the full name of each committer to each detected copyright notice. We achieve this by checking whether the full name is included in the detected copyright notice or not. Note that this check is not case-sensitive. If we find the full name of one committer in the detected copyright notice, we determine it as the holder of this copyright notice. At the same time, the type of this holder is determined as individual. For the remaining copyright notices, we try to match them to the organization names in all lists in the organization dictionary. If an organization name is matched, we determine the index - the uniform identifier refers to the list containing this organization name - as the holder. At the same time, the type of this holder is determined as organization.

Table 4 A part of the built organization dictionary.

Index	List
ibm	IBM Corporation IBM Corp. International Business Machines Corp.
amd	Advanced Micro Devices, Inc AMD, Inc
ti	Texas Instruments, Inc. TI, Inc

After these two matches, we try to manually identify the holders and their types for the remaining copyright notices. If the manually identified organization name has got a uniform identifier in the organization dictionary, we determine this uniform identifier as the holder. Meanwhile, we add this manually identified organization name into the list that the uniform identifier refers to. If the manually identified organization name has not got a uniform identifier in the organization dictionary, we create a new uniform identifier as the index and add this manually identified organization name into the list the newly created uniform identifier refers to. Finally, we build a holder dataset, for each source file in which the holders and the information about their related copyright notices and their types (individual or organization) are summarized.

During this process, we successfully build an organization dictionary at the same time. Table 4 shows a part of the built organization dictionary constructed during the analysis of the Linux kernel.

4.3 Analysis Method

To answer RQ1, we detect the copyright inconsistencies based on the built committer dataset and holder dataset. Based on the definitions in Sect. 3.1, we detect two types of copyright inconsistencies respectively - the committer-not-holder inconsistency and the holder-not-committer inconsistency.

To achieve our goals, we propose two definitions of committer for the detection of two types of copyright inconsistency. *General committers* refer to all committers who once committed source code. This definition will be used in the detection of the holder-not-committer inconsistency, which ensures that the holder-not-committer inconsistency can only be detected when we can not find any information about the holder in the committer dataset. *Core committers* refer to the committers who contributed more than a mini-

mum threshold percentage of contribution. Setting a minimum threshold percentage of the contribution could exclude the minor committers who only do some simple work, and determines the core committer from a general committer. We will set this minimum percentage as 14.9% because it is the least percentage of source code at the token granularity of the contributor who committed the highest percentage of the source code in the sample dataset. This definition will be used in the detection of the committer-not-holder inconsistency.

We first detect the holder-not-committer inconsistency. For each holder in the holder dataset, we check whether this holder is a name of any general committer or an organization that any general committer belongs to. A holder-not-committer inconsistency is detected if a holder is neither a name of any general committer or an organization that any general committer belongs to.

We then detect the committer-not-holder inconsistency. For each core committer in the committer dataset, we check whether its name or organization is recorded in the holder dataset or not. A committer-not-holder inconsistency is detected if neither a core committer's name nor its organization is recorded in the holder dataset.

To answer RQ2, we try to find the reasons behind the occurrence of the holder-not-committer inconsistency and the committer-not-holder inconsistency respectively. For each inconsistency, we manually check the commit logs and the comments in the source code of the source files to find out the reasons. A reason is determined only when it is explicitly recorded in the commit logs or the comments in source code.

5. Results

In this section, we report the results and have a discussion on the results to address research questions in our empirical study in Sect. 4.

5.1 RQ1: How Prevalent Are Copyright Inconsistencies?

In this research question, we analyze to what extent copyright inconsistencies exist in source files in the Linux kernel.

As can be seen from Table 5, 262 source files are detected as having copyright inconsistencies, accounting for 63.3% of all 414 source files. As a popular and well maintained OSS project, the copyright ownership of the Linux kernel should be clear and well maintained as well. The general image of the Linux communities is that there is not too much copyright inconsistency in the Linux kernel. An ideal situation is expected that there is no inconsistency. However, this result is out of our expectation and do not accord with the general image of the communities, which suggests that copyright inconsistencies are prevalent in the Linux kernel.

We can see that 134 source files are detected as having the holder-not-committer inconsistency, accounting for 32.4% of all 414 source files in sample dataset, and 51.1% of 262 source files detected as having any type of inconsistencies respectively. We can also see that 229 source files are detected as having the committer-not-holder inconsistency, accounting for 55.3% of all 414 source files in sample dataset, and 87.4% of 262 source files detected as having any type of inconsistencies respectively. It can be noticed that the committer-not-holder inconsistency is more prevalent than the holder-not-committer inconsistency.

Table 5 The number of source files with different types of copyright inconsistencies and the ratios. Ratio to ① means the ratio of the source files detected as having this type of inconsistency to all source files in the sample dataset. Ratio to ④ means the ratio of the source files detected as having this type of inconsistency to the source files detected as having any type of inconsistency.

No	Inconsistency	#Files	Ratio to ①	Ratio to ④
①	sample dataset	414	100.0%	
②	holder-not-committer	134	32.4%	51.1%
③	committer-not-holder	229	55.3%	87.4%
④	any type of inconsistency	262	63.3%	100.0%
⑤	both two types	101	24.4%	38.5%

It is also easy to know that 101 source files are detected as having both of two types, accounting for 24.4% of all 414 source files in sample dataset, 38.5% of 262 source files detected as having any type of inconsistencies, 75.3% of 134 source files detected as having the holder-not-committer inconsistency, and 44.1% of 229 source files detected as having the committer-not-holder inconsistency respectively. The result suggests that if the holder-not-committer inconsistency exists in a source file, there is also a high possibility of the occurrence of the committer-not-holder inconsistency. Oppositely, if the committer-not-holder inconsistency exists in a source file, there is no such high possibility of the occurrence of the holder-not-committer inconsistency. It may be a possible reason that for the source files detected as having the holder-not-committer inconsistency, the committers added others' copyright notices. Another possible reason may be that after a long time of software evolution, a lot of new source code is committed, and the new source code replaced the old source code committed by the committers before. We will investigate the reasons in RQ2.

As an answer to RQ1, copyright inconsistencies are prevalent in the Linux kernel, among which, the committer-not-holder inconsistency is more prevalent than the holder-not-committer inconsistency.

5.2 RQ2: What Caused the Copyright Inconsistencies?

We aim to find the reasons why copyright inconsistencies happened. To achieve this goal, for the holder-not-committer inconsistency, we manually check the commit logs and the comments in the source code of all 134 source files detected as having the holder-not-committer inconsistency. If a reason is explicitly recorded in a sentence, we note down that sentence. After that, we categorized all sentences to summarize the reasons.

For the committer-not-holder inconsistency, we select

some source files detected as having the committer-not-holder inconsistency as the target. These source files should also satisfy the condition that no holder-not-committer inconsistency is detected. We end up with 128 source files. For these files, all holders are the committers, so we plan to investigate the other committers who are not holders to find why they are not recorded in copyright notices. Different from the holder-not-committer inconsistency, holders who did not add their copyright notices usually did not explain their reasons explicitly. So we try to find some hints about why holders did not add their copyright notices no matter they are explicit or not. We then categorized all sentences to summarize the reasons as well.

Table 6 shows the summarized reasons why holder-not-committer inconsistencies happened. Among them, *code reuse* and *refactoring* are two common activities in software development. It is possible that copyright notices are not well managed during these activities.

To our surprise, *affiliation change* - the change of the companies or organizations the developers belong to - plays an important role in the occurrence of the copyright inconsistency, which reveals that the management of copyright notices may be overlooked by the developers as well. The "affiliation change" is found when we observed that although the developer's affiliation identified by e-mail address is not consistent with the one in the copyright notice currently, this developer's affiliation may be consistent with the one in the copyright notice before. Specifically, we observed four cases: (1) The developer switches to a different company or organization. (2) The developer belongs to more than one company or organization. (3) The company or organization is merged into another one. (4) The developer uses a personal e-mail address. All these cases can be found by checking the commit logs, the comments in the source code of the source files, and the profiles of the developers. We also try to search the related information on the Internet (e.g. the information of the developer on LinkedIn[†], the information of the company or organization in Wikipedia[††], etc.) to endorse our findings. Note that we do not explore the reasons why "affiliation change" happens and whether possible legal risks exist. Developers may just forget to update the copyright notices and the legal risks also vary across different countries or regions. However, we do observe that "affiliation change" is an important reason causing copyright inconsistency and needs to be noticed.

Support function and *others' contributions* are two reasons why developers add others' copyright notices when they committed source code. Another interesting case is *All replaced*, which is the situation that the source code the holder committed is totally replaced by the source code committed by the committers later, but the copyright notice is retained. *All replaced* also suggests that copyright notices are not well managed in the Linux kernel. We draw a conclusion that *code reuse*, *affiliation change*, *refactoring*, *support function*, and *others' contributions* are the main reasons why the holder-not-committer inconsistency occurred.

Table 6 The reasons why holder-not-committer inconsistencies happened in the Linux kernel version 4.14.

Categorization	#Source files	Proportion
Code reuse	37	27.6%
Affiliation change	19	14.2%
Refactoring	14	10.4%
Support function	13	9.7%
Others' contributions	13	9.7%
Typo	5	3.7%
All replaced	2	1.5%
None	31	23.2%
Total	134	100.0%

Table 7 The reasons why committer-not-holder inconsistencies happened in the Linux kernel version 4.14.

Categorization	#Source files	Proportion
Code reuse	30	23.4%
Affiliation change	12	9.4%
Refactoring	8	6.3%
Others' contributions	7	5.5%
Typo	1	0.8%
None	70	54.6%
Total	128	100.0%

Table 7 shows the summarized reasons why committer-not-holder inconsistencies happened. The results are similar to the results of holder-not-committer inconsistency. *Code reuse*, *affiliation change*, *refactoring*, *support function*, and *others' contributions* are still the main reasons why the committer-not-holder inconsistency occurred.

Based on these findings, the following practical suggestions may help practitioners: (1) When developers reuse source code, the provenance of reused source code should be recorded properly as well for the traceability of the copyright notice. (2) The list of contributors should be well maintained as an evidence of copyright ownership. (3) The communities should propose a set of practical guidelines for managing the copyright notices. For example, when a copyright notice is added, modified, or deleted, the reason and the coverage of influence should be recorded. (4) The copyright ownership should be ascertained with a finer granularity such as line level or token level. The related tools are needed to be developed.

As an answer to RQ2, we draw a conclusion that *code reuse*, *affiliation change*, *refactoring*, *support function*, and *others' contributions* are the main reasons why copyright inconsistency occurred.

6. Threats to Validity

This section discusses the threats to the validity of our research. Threats to construct validity concern the relationship between theory and outcome, and relate to possible

[†]https://www.linkedin.com/

[††]https://en.wikipedia.org/

measurement imprecision when extracting data we used in this study. In mining the repositories to build the committer dataset, we first rely on `cregit` to summarize the names of the committers and their contributions. `cregit` measures the contribution in terms of tokens. Compared with the method measuring the contribution in terms of lines before, `cregit` is more precise and could avoid the case that developers doing simple code change are seen as the owner of the total lines. However, the precision of `cregit` is not proved by a large-scale test. To limit this problem, we randomly select some source files to check the precision of `cregit` at the same time when we do the empirical study. Another threat in using `cregit` is its method of merging the multiple different accounts owned by the same committer. `cregit` merges them based on the full name of the committer. This method will be ineffective for the case that a committer uses more than one name to commit source code to a single source file. However, this defect does not affect the results because we do not observe this case in our empirical study.

Another threat in building the committer dataset is that we identify the organizations the committers belong to by checking the domain of their e-mail addresses. But the committers possibly use their personal e-mail addresses. Also, the committers may not change their e-mail addresses in Github when they change their organizations. For these cases, we can not rightly identify their organizations. Considering that we also track the change of their organizations in answering the RQ2, this threat might have an impact on our empirical study as well.

In mining the repositories to build the holder dataset, we first rely on FOSSology to extract the copyright notices. Although we manually check the copyright notices detected by FOSSology to remove the wrongly detected ones, FOSSology could have missed some copyright notices. We plan to use other methods or tools to do the detection in the future.

Another case worthwhile of being discussed is the minimum threshold percentage of contribution to define the committer used to detect the committer-not-holder inconsistency. We have set this minimum threshold percentage as 14.9%. The percentage of source files detected as having the committer-not-holder inconsistency will increase if we do not set it. In our calculation, if we would lift the threshold, 87.9% of all source files are detected as inconsistent, which proves the effectiveness of the minimum threshold percentage in excluding the minor committers who only do some simple work. Some studies set the minimum threshold percentage as 5% to exclude the minor contributors [10], [11]. To what extent the contribution is needed to become a qualified contributor or state the copyright ownership is still a complicated problem under discussion.

Another threat requiring consideration is the effectiveness of the sampling. To achieve an accurate result, a lot of manual works are included in our proposed method to detect copyright inconsistency, for which we have introduced sampling. To validate the effectiveness of the sampling and the manual works, we have conducted a comparison experiment. We first designed a fully-automatic method without manual works to detect copyright inconsistency and then used it to detect copyright inconsistencies targeting all 38,932 source files in the Linux kernel and 500 source files the sample dataset respectively. For all source files in the Linux kernel, 19,261 source files out of 38,932 total source files are detected as having hold-not-committer inconsistency, accounting for 49.5%, and 24,143 source files are detected as having committer-not-hold inconsistency, accounting for 62.0%. For the sample dataset, 228 source files out of 500 files are detected as having hold-not-committer inconsistency, accounting for 45.6%, and 309 source files are detected as having committer-not-hold inconsistency, accounting for 61.8%. It is easy to find that the results are similar, which suggests that the sampling is effective. Also, this sampling method based on population size, confidence interval, and confidence level is well established. It is first proposed by Krejcie and Morgan in 1970 [12], and widely used and proven by other related works [13]–[15]. Then we have compared the results detected by the proposed method in Table 5 and the results detected by the newly designed fully-automatic method targeting the sample dataset. The proportions detected by the proposed method are smaller. Because of the manual works in the proposed method, the copyright inconsistencies detected by the proposed method are all actual ones while the copyright inconsistencies detected by the newly designed fully-automatic method are not. The results suggest that the manual works are effective and could improve accuracy.

Threats to internal validity concern factors internal to the study that could impact our results. Such a kind of threat does not affect exploratory study like the one in this paper. The only case worthwhile of being discussed is our answering to RQ2, where we classify the reasons manually based on our knowledge.

Threats to external validity are related to the ability to generalize the finding in our study. Our empirical study is only conducted on the Linux kernel. Linux kernel is the most popular and successful open-source operating system kernel, receiving contribution from 13,500 developers from more than 1,300 organizations. These features make Linux kernel a suitable target to study copyright inconsistencies. But according to the different requirements about the copyright, other open-source projects may have different results. We agree that it is necessary to replicate our empirical study on different projects.

Another case worthwhile of being discussed is the generalization of our findings in RQ2. To address this issue, we have repeated our experiment targeting another version of the Linux kernel and checked whether or not we can achieve similar results. The new experiment targeted version 5.80 of the Linux kernel, from which we randomly selected 500 source files to construct the sample dataset. After removing the source files once moved or renamed, we ended up with a sample dataset of 390 source files. We then repeated the detection. Among these 390 source files, 126 source files are detected as having hold-not-committer inconsistency,

Table 8 The reasons why holder-not-committer inconsistencies happened in the Linux kernel version 5.80.

Categorization	#Source files	Proportion
Code reuse	38	30.2%
Affiliation change	21	16.7%
Refactoring	12	9.5%
Support function	17	13.5%
Others' contributions	8	6.3%
Typo	2	1.6%
None	28	22.2%
Total	126	100.0%

Table 9 The reasons why committer-not-holder inconsistencies happened in the Linux kernel version 5.80.

Categorization	#Source files	Proportion
Code reuse	25	22.5%
Affiliation change	9	8.1%
Refactoring	6	5.4%
Others' contributions	2	1.8%
None	69	62.1%
Total	111	100.0%

accounting for 32.3%, while 205 source files are detected as having committer-not-hold inconsistency, accounting for 52.6%. Among the 205 source files where committer-not-hold inconsistency are detected, 111 source files are detected as having only committer-not-hold inconsistency and no hold-not-committer inconsistency. Then we have used the same method to find reasons behind the occurrence of the copyright inconsistency. Table 8 and Table 9 show the results. The similar results suggest the ability to generalize the finding in RQ2.

7. Related Work

7.1 Software Ownership

Some studies in software engineering investigated software ownership. Girba et al. built the ownership based on the percentage of source code lines modified by contributors [16]. Tsikerdekis et al. proposed a code contribution ranking algorithm to build the ownership by tracking the survival of individual characters [17]. Bird et al. explored the effects of ownership on software quality [10]. Compared with their works, we use a more accurate and reasonable measure - token - to measure contributions. Especially, different from the above works, we also included copyright notice into consideration. Our work opens up a new way to study software ownership.

7.2 OSS Contributor

There are some studies that devoted to investigate OSS contributors and their contributions. German et al. studied the committers of the PostgreSQL project and found that apart from the core team, a large number of contributors sent source code patches to the project [18]. Hindle et al. discovered that the large commits including a large number of files are related to license or copyright owners [19]. Hammad et al. proposed two measures to measure the contribution of software developers in the evolved structural design of software systems [20]. Different from their works, we discussed the discrepancy between the contributors' contributions and the recorded copyright notices. Our work creates a possibility of importing the existing works on OSS Contributor into the software copyright management of the FOSS projects.

7.3 Software License

There are some studies that devote to identify licenses [5], [21]. Based on these studies, some researchers analyzed software licenses in open source projects and revealed some license issues. German et al. proposed a method to understand licensing compatibility issues in software packages [22]. Wu et al. proposed an approach to find license inconsistencies in similar files [23]. By investigating the revision history of these files, they summarized the factors that caused these license inconsistencies and tried to decide whether they are legally safe or not. Studies on software license are also closely related to this work. Many studies in software engineering investigated software license. However, to solve the legal risks in reusing FOSS, only the studies on software license are not sufficient. This work is a supplement to works on software license by studying the software ownership.

8. Conclusion and Future Work

In this paper, we first proposed the issue of copyright inconsistency, and then we defined copyright inconsistency and categorized different types of it. After that, we conducted an empirical study on the Linux kernel to study the prevalence of copyright inconsistency. To the best of our knowledge, this study is the first in this field to address this issue. We observed that the copyright inconsistency is prevalent in the Linux kernel. It suggests that the copyright notices recorded in the source files do not always reflect the actual contributors. To find how copyright inconsistency happens, we had a deeper look at the commit logs and the comments in source code to find the reasons why the copyright inconsistencies happened. We found that *code reuse*, *affiliation change*, *refactoring*, *support function*, and *others' contributions* are the main reasons.

The proposed method and results of this work can be reused in the following aspects: (1) The proposed method of detecting copyright inconsistency can be reused in other OSS projects to study the situation of copyright inconsistency in them. (2) Our findings on the prevalence of copyright inconsistency and the reasons causing them can be used as a benchmark to study the difference in the copyright-related issue among different OSS projects. (3) Our findings

also provide a new perspective to study software development management, code reuse, and the organizational culture in OSS projects. (4) The datasets constructed in this work can be reused in other highly related works such as the participation of developers or companies in OSS projects and the collaboration between them.

In our future work, we will make some guidelines for developers to help them in dealing with copyright notices. We also aim to find a solution to manage the copyright notices in the FOSS projects.

References

[1] M. Larabel, "The linux kernel enters 2020 at 27.8 million lines in git but with less developers for 2019," Web page at linux.com, https://www.phoronix.com/scan.php?page=news_item&px=Linux-Git-Stats-EOY2019, Jan. 2020.
[2] T. Golder and A. Mayer, "Whose ip is it anyway?," Journal of Intellectual Property Law & Practice, vol.4, no.3, pp.165–175, 2009.
[3] H. Meeker, "Patrick mchardy and copyright profiteering," Web page at opensource.com, https://opensource.com/article/17/8/patrick-mchardy-and-copyright-profiteering, Aug. 2017.
[4] H. Welte, "Report from the geniatech vs. mchardy gpl violation court hearing," Web page at gnumonks.org, https://laforge.gnumonks.org/blog/20180307-mchardy-gpl/, March 2018.
[5] R. Gobeille, "The FOSSology project," 2008 International working conference on Mining software repositories (MSR2008), pp.47–50, 2008.
[6] M.D. Penta and D. German, "Who are source code contributors and how do they change?," 2009 16th Working Conference on Reverse Engineering, pp.11–20, Jan. 2009.
[7] Oracle, "Oracle contributor agreement - version 1.7.1," Web page at oracle.com, https://www.oracle.com/technetwork/community/oca-486395.html#list, 2019.
[8] J. Corbet and G. Kroah-Hartman, "2017 linux kernel development report," A Publication of The Linux Foundation, 2017.
[9] D.M. German, B. Adams, and K. Stewart, "cregit: Token-level blame information in git version control repositories," Empir. Softw. Eng., vol.24, issue 4, pp.2725–2763, 2019.
[10] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," Proc. 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11, pp.4–14, 2011.
[11] M. Foucault, J.-R. Falleri, and X. Blanc, "Code ownership in open-source software," Proc. 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14, pp.1–9, 2014.
[12] R.V. Krejcie and D.W. Morgan, "Determining sample size for research activities," Educ. Psychol. Meas., vol.30, no.3, pp.607–610, 1970.
[13] H. Hata, C. Treude, R.G. Kula, and T. Ishio, "9.6 million links in source code comments: purpose, evolution, and decay," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp.1211–1221, IEEE, 2019.
[14] Z. Gao, C. Bird, and E.T. Barr, "To type or not to type: quantifying detectable bugs in javascript," 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp.758–769, IEEE, 2017.
[15] A.D. Veiga, "A cybersecurity culture research philosophy and approach to develop a valid and reliable measuring instrument," 2016 SAI Computing Conference (SAI), pp.1006–1015, IEEE, 2016.
[16] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," 8th international workshop on principles of software evolution (IWPSE'05), pp.113–122, 2005.
[17] M. Tsikerdekis, "Persistent code contribution: A ranking algorithm for code contribution in crowdsourced software," Empir. Softw. Eng., vol.23, no.4, pp.1871–1894, 2018.
[18] D.M. German, "A study of the contributors of PostgreSQL," Proc. 2006 international workshop on Mining software repositories - MSR '06, pp.163–164, 2006.
[19] A. Hindle, D.M. German, and R. Holt, "What do large commits tell us? a taxonomical study of large commits," Proc. 2008 international workshop on Mining software repositories - MSR '08, pp.99–108, 2008.
[20] M. Hammad, M. Hammad, H. Bani-Salameh, and E. Fayyoumi, "Measuring developers' design contributions in evolved software projects," Journal of Software, vol.9, no.12, pp.3005–3011, 2014.
[21] D.M. German, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," Proc. IEEE/ACM international conference on Automated software engineering - ASE '10, pp.437–446, 2010.
[22] D.M. German, M.D. Penta, and J. Davies, "Understanding and auditing the licensing of open source software distributions," 2010 IEEE 18th International Conference on Program Comprehension, pp.84–93, 2010.
[23] Y. Wu, Y. Manabe, T. Kanda, D.M. German, and K. Inoue, "A method to detect license inconsistencies in large-scale open source projects," 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp.324–333, 2015.

Shi Qiu received his B.E. degree of software engineering from Jilin University in 2013 and his M.E. degree of information science and technology from Osaka University in 2018. At present, he is a Ph.D. student in the Graduate School of Information Science and Technology at Osaka University. His research interests include mining software repositories, software license/copyright analysis, and software ecosystem.

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

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