Large-scale Analysis of Software Reuse for Code and License Changes

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2019

Yuhao WU

Abstract

Code reuse is a very common practice in software engineering. When performed in a correct way, code reuse can help developers create software products with higher quality more efficiently. Although code reuse is beneficial in many aspects, there are also several issues that we need to take special care of. One important aspect is software license, without which source code cannot be reused legally. Another aspect is the efficiency or barriers during the process code reuse.

In the first part of this dissertation, we deal with the issue of software license. Software license is a written text that grants the permissions of reusing and redistributing the software to its users. Removing or modifying the license statement by re-distributors will result in the inconsistency of license with its ancestor, and may potentially cause license infringement. However, in our study we have encountered cases where multiple source files that have the same source code but are under different licenses. Therefore, we describe and categorize different types of license inconsistencies and propose a method to detect them. Then we applied this method to Debian 7.5 and a collection of 10,514 Java projects on GitHub and present the license inconsistency cases found in these systems. With a manual analysis, we summarized various reasons behind these license inconsistency cases, some of which imply potential license infringement and require attention from the developers. This analysis also exposes the difficulty to discover license infringements, highlighting the usefulness of finding and maintaining source code provenance.

In the second part of this dissertation, we deal with the barriers during the process of code reuse. Although code reuse is a common practice, the process is not fully studied: how often and why is the source code changed during code reuse, what hinders code reuse and how can we improve it? In order to address these issues, we conduct an empirical study on code reuse from Stack Overflow, a question and answer (Q&A) platform for software developers. In this study, we first conduct an exploratory study on 289 files from 182 open source projects, which contain source code that has an explicit reference to a Stack Overflow post. We found that code modification during code reuse is a frequent action. Meanwhile, developers also

write (re-implement) source code from scratch based on the idea from Stack Overflow. To further understand the barriers of reusing code and to obtain suggestions for improving the code reuse process on Q&A platforms, we conducted a survey with 453 open source developers who are also on Stack Overflow. We found that the top 3 barriers that make it difficult for developers to reuse code from Stack Overflow are: (1) too much code modification required to fit in their projects, (2) incomprehensive code, and (3) low code quality. We summarized and analyzed all survey responses and we identified that developers suggest improvements for future Q&A platforms along the following dimensions: code quality, information enhancement & management, data organization, license, and the human factor. Our findings can be used as a roadmap for researchers and developers to improve code reuse.

List of Publications

Major Publications

1. Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, Katsuro Inoue. "How Do Developers Utilize Source Code from Stack Overflow?" The 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018), Journal-First, Lake Buena Vista, Florida, November 2018.

2. Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, Katsuro Inoue. "How Do Developers Utilize Source Code from Stack Overflow?" Empirical Software Engineering Journal, 2018 (to appear).

3. Yuhao Wu, Yuki Manabe, Daniel M. German, Katsuro Inoue. "How Are Developers Treating License Inconsistency Issues? A Case Study on License Inconsistency Evolution in FOSS Projects." The 13th International Conference on Open Source Systems (OSS 2017), pp. 69-79, Buenos Aires, Argentina, May 2017.

4. Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. German, Katsuro Inoue. "Analysis of License Inconsistency in Large Collections of Open Source Projects." Empirical Software Engineering Journal, Vol.22, No.3, pp.1194-1222, 2017.

5. Yuhao Wu, Yuki Manabe, Tetsuya Kanda, Daniel M. German, Katsuro Inoue. "A Method to Detect License Inconsistencies in Large-Scale Open Source Projects." in Proceedings of the 12th Working Conference on Mining Software Repositories (MSR 2015), pp.324-333, Florence, Italy, May 2015.

Related Publications

1. Anfernee Goon, Yuhao Wu, Makoto Matsushita, Katsuro Inoue. "Evolution of Code Clone Ratios throughout Development History of Open-

Source C and C++." 11th International Workshop on Software Clones (IWSC 2017), pp. 47-53, Klagenfurt, Austria, February 2017.

2. Anfernee Goon, Yuhao Wu, Makoto Matsushita, Katsuro Inoue. "Analysis of Code Clone Ratios over Version Evolution in Open-Source Projects Written in C and C++." Software Engineer Symposium (2016), pp. 255-256, 2016.

3. Buford Edwards III, Yuhao Wu, Makoto Matsushita, Katsuro Inoue. "Estimating Code Size After a Complete Code-Clone Merge." Software Engineering Research Report (SE), 2016(3), pp. 1-8.

Acknowledgement

First of all, I am deeply grateful to my supervisor: Professor Katsuro Inoue, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my Ph.D study.

I would like to thank my co-supervisors: Professor Daniel M. German (affiliated with University of Victoria) and Assistant Professor Yuki Manabe (Kumamoto University). They gave me a lot of motivating research ideas and guided me through the difficulties during the research.

Besides my supervisors, I would like to thank the rest of my thesis committee: Professor Shinji Kusumoto and Professor Fumihiko Ino, for their insightful comments and encouragement.

My sincere thanks goes to my collaborators: Dr. Shaowei Wang (Queen's University), Assistant Professor Cor-Paul Bezemer (University of Alberta) and Professor Ahmed E. Hassan (Queen's University), for their kind support during my internship in Queen's University. They gave me a lot of valuable comments and suggestions on my research and paper writing, which I believe will definitely be very helpful during the rest of my life.

I am grateful to Dr. Akira Mori, who provided me with a lot of wonderful research ideas and enlightening comments during my internship in National Institute of Advanced Industrial Science and Technology (AIST).

I would like to thank Professor Makoto Matsushita, Assistant Professor Tetsuya Kanda, Professor Takashi Ishio (Nara Institute of Science and Technology), Assistant Professor Raula Gaikovina Kula (Nara Institute of Science and Technology), Assistant Professor Eunjong Choi (Nara Institute of Science and Technology), Professor Norihiro Yoshida (Nagoya University), Dr. Xin Yang, Professor Ali Ouni (University of Quebec), Professor Coen De Roover (Vrije Universiteit Brussel), Dr. Leon Moonen (Simula) and Ms. Kate Stewart (The Linux Foundation), for their support and advices.

I would like to thank all the members in my laboratory for creating such a wonderful environment for studying and researching. My special thanks

goes to our lab secretary: Ms. Mizuho Karube, for her continuous support and encouragement in the past 5 years.

I would also like to thank all the participants in our web survey of our study, for providing plentiful valuable responses.

Last but not least, I would like to thank my family members for supporting me spiritually throughout writing this dissertation and my life in general.

Contents

1 Introduction **1**
- 1.1 Software License . . . 1
  - 1.1.1 Free and Open Source License . . . 2
  - 1.1.2 License Inconsistency . . . 2
  - 1.1.3 A Motivating Example . . . 3
- 1.2 Software Reuse . . . 4
  - 1.2.1 Question and Answer Platforms . . . 4
  - 1.2.2 A Motivating Example . . . 5
- 1.3 Contributions of the Dissertation . . . 5
  - 1.3.1 Software License . . . 5
  - 1.3.2 Source Code . . . 6
- 1.4 Overview of the Dissertation . . . 6

2 Analysis of License Inconsistency in Large Collections of Open Source Projects **9**
- 2.1 Introduction . . . 9
- 2.2 License Inconsistencies . . . 11
  - 2.2.1 Definition . . . 13
  - 2.2.2 Example . . . 13
  - 2.2.3 Categorization . . . 14
- 2.3 Method to Detect License Inconsistencies . . . 15
  - 2.3.1 License Inconsistency Metrics . . . 17
  - 2.3.2 Method of Detecting License Inconsistencies . . . 18
  - 2.3.3 Example . . . 18
- 2.4 Empirical Study . . . 20
  - 2.4.1 Empirical Study on Debian 7.5 . . . 21
  - 2.4.2 Empirical Study on Java Projects . . . 28
  - 2.4.3 Discussion of the Results . . . 32
  - 2.4.4 Answering RQs . . . 33
- 2.5 Discussion . . . 34
  - 2.5.1 Improvement of the Method . . . 34
  - 2.5.2 What Appears to Be a Copy Might Not Be a Copy . 35

2.5.3 Changes Were Made Under the Permission of Copyright Owner . . . 36
2.5.4 An Attempt in Measuring the Recall . . . 36
2.6 Threats to Validity . . . 37
2.7 Related Work . . . 38
2.8 Conclusion of This Chapter . . . 39

3 How Do Developers Utilize Source Code from Stack Overflow? 41
3.1 Introduction . . . 41
3.2 Background & Related Work . . . 43
3.2.1 Leveraging Knowledge from Stack Overflow . . . 43
3.2.2 Understanding the Quality of Posts on Stack Overflow 44
3.2.3 Source Code Reuse from Stack Overflow . . . 45
3.2.4 Code Licensing on Stack Overflow . . . 46
3.3 Research Questions & Data Collection . . . 47
3.3.1 Research Questions . . . 47
3.3.2 Data Collection . . . 48
3.4 An Exploratory Study of Source Code Reuse from Stack Overflow in Open-Source Projects . . . 51
3.4.1 RQ1: To What Extent Do Developers Need to Modify Source Code From Stack Overflow in Order to Make It Work in Their Own Projects . . . 51
3.4.2 RQ2: From Which Part of the Stack Overflow Post Does the Reused Source Code Come? . . . 59
3.5 A Survey on Code Reuse from Stack Overflow . . . 62
3.5.1 RQ3: What Are the Preferences of Developers When It Comes to Reusing Code? . . . 63
3.5.2 RQ4: Is Code License a Barrier for Code Reuse for Developers? . . . 65
3.6 A Roadmap for Next-Generation Q&A platforms . . . 66
3.6.1 Suggestions on Code Quality . . . 69
3.6.2 Suggestions on Information Enhancement & Management . . . 72
3.6.3 Suggestions on Data Organization . . . 74
3.6.4 Suggestions on Code License . . . 76
3.6.5 Suggestions on the Human Factor . . . 78
3.6.6 Other Suggestions . . . 80
3.7 Threats to Validity . . . 80
3.8 Conclusion of This Chapter . . . 82

4 Conclusion and Future Work 85
4.1 Conclusion . . . 85
4.2 Future Directions . . . 86

List of Figures

2.1 Hierarchy of a project and the license of each source file. Note that the foo.c file in `Pkg1` was imported to `Pkg2` with the license changed to GPL-3.0+; The foo.c in `Pkg3` contains totally different source code than the one in `Pkg1`, and was imported to `Pkg4` with its name changed to foo100.c and license removed. 19
2.2 Hierarchy of the grouped files. 19

3.1 An example of a question and its accepted answer on Stack Overflow. 45
3.2 An overview of our data collection of the exploratory study. 49
3.3 The distribution of the studied Stack Overflow links over the five programming languages. 50
3.4 Distribution of the software engineering experience of the participants in years. 50
3.5 Distribution of the types of projects that the participants are working on. 51
3.6 The distribution of each type of source code utilization for each of the studied programming languages. 54
3.7 Comparison of frequency of reusing and reimplementing source code. 63
3.8 Participants' awareness of the licenses of Q&A platforms. . 65
3.9 Participants' opinion about license compatibility between Q&A platforms and their projects. 65
3.10 Importance of having more information on license. 66

List of Tables

2.1 Strategies to decide whether a certain type of license inconsistency exists in a group. 17
2.2 License list of the selected files from the example project. . 20
2.3 List of the license inconsistency metrics for each file group in the example project. 20
2.4 Main characteristics of Debian 7.5. 21
2.5 Breakdown of number of groups and files for each type in analyzing Debian 7.5. 21
2.6 Partial list of the license inconsistency metrics for each file group in detecting Debian 7.5. 22
2.7 Number of different types of license inconsistencies and their proportion in Debian 7.5. Note that one group may contain more than one inconsistency types, so that the total percentage can exceed 100%. 22
2.8 Example of *LAR* inconsistency, in getopt.c 22
2.9 License list of group 6645 of obstack.c where *LUD* exists. . . 24
2.10 License list of group 52662 of getopt.c where *LC* and *LAR* exist. 24
2.11 The count and percentage of each category for the 25 investigated license inconsistency cases. 26
2.12 Main characteristics of Java projects cloned from GitHub. . 29
2.13 Number of groups and files in each group in analyzing Java projects. 29
2.14 Number of different types of license inconsistencies and their proportion in Java projects. 29
2.15 The count and percentage of each category for the 17 investigated license inconsistency cases in the Java projects. . . . 30
2.16 Comparison of two methods on Debian 7.5. 35
2.17 Comparison of two methods on Java projects. 35

3.1 The identified types of source code utilization from Stack Overflow. 53
3.2 Where does the reused source code come from? 59

3.3 Reasons for choosing reimplementing over reusing source code. (Multi-selection allowed, hence the sum of the percentages is larger than 100%.) 64
3.4 The categorization of code quality suggestions — 64 out of 183 (35.0%). 68
3.5 The categorization of information enhancement & management suggestions — 43 out of 183 (23.5%). 71
3.6 The categorization of data organization suggestions — 21 out of 183 (11.5%). 73
3.7 The categorization of code license suggestions — 23 out of 183 (12.6%). 75
3.8 The categorization of human factor suggestions — 19 out of 183 (10.4%). 77
3.9 The categorization of other suggestions — 13 out of 183 (7.0%). 79

Chapter 1

Introduction

Software is playing a more and more important role nowadays. It supports our daily life in all aspects: mobile phones, medical devices and aircrafts all rely on software. However, many incidents have proved that, software errors can result into financial loss or even people's death [88]. There are several approaches to mitigate the influence of software errors. Among them, code reuse is proved to be an efficient and effective way to reduce software errors while improving code quality [14, 17, 51, 61, 87]. On one hand, code reuse saves the development time comparing to writing the software from scratch, reducing the cost for the company [8]. On the other hand, the reused source code is often widely tested by many other developers, thus resulting in a software with higher quality [14, 49].

Despite the benefits, code reuse also comes with many risks: reusing source code with an incompatible license may result into license violation; integrating source code that you do not understand may reduce the readability and maintainability of the target software. In this dissertation, we deal with software license issues and the barriers during code reuse. We introduce each of them in the following sections respectively.

1.1 Software License

Software licensing and licensing of digital information in general create a regime of information governance for the Internet and beyond [58]. Generally speaking, there are two types of software licenses: free and open source software (FOSS) license and closed source license. FOSS licensing schemes permit users to access both the source code and object code of a particular computer program. In contrast, conventional or closed source licensing schemes typically permit access only to the object code, preventing manipulations of the underlying program itself. Since our research addresses issues of the source code level reuse, we will focus on FOSS license in this dissertation.

1.1.1 Free and Open Source License

With the growth of free and open source software, code reuse from the FOSS software is playing a more and more important role in project development [81]. FOSS license allows the software to be freely used (as in freedom), modified, and redistributed (in modified or unmodified form) by anyone, as long as the conditions of its license are satisfied. Broadly speaking, there are two types of FOSS licenses: *restrictive* and *permissive* licenses. Restrictive licenses require the derivative work to be under the same license as the work they based on, while permissive licenses allow reusers to redistribute the derivative work under other licenses that the reusers choose [80].

For example, GPL is a typical *restrictive license*. Under Section 5 of GPL-3.0, it includes such statement: *"You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy"* [26]. Other restrictive licenses include: LGPL, AGPL, etc. On the other hand, BSD is a typical *permissive license*. For example, BSD-2-Clause has no restrictions on what license the derivative work should use; the only requirements are: 1) copyright notice been included in the source code, and 2) binary form reproducing the copyright notice [45]. Other permissive licenses include: MIT, Apache-2.0, etc.

Although FOSS software can usually be reused without any cost, developers should be very careful about the terms included in the license, otherwise license issues may occur.

License compatibility is one the issues where source code/packages under different licenses are mixed together while the license terms conflict with each other, making them incompatible. German *et al.* [33] have proposed a method to help understand licensing issues and performed an empirical study which showed the existence of license incompatibilities issues in Fedora-12.

Another license issue which lacks the attention from researchers is the license inconsistency issue. We discuss this issue in the following section.

1.1.2 License Inconsistency

In our research, license inconsistency is defined as: two source files with same source code but having different license statements. Note that license inconsistency is different from license incompatibility. License incompatibility is the problem that two standalone software artifacts cannot be integrated together because of license terms confict with each other; while license inconsistency is the situation that two source files having the same source code, but are with different licenses. License inconsistency can occur when a reuser copy and paste the file while modifying the license of the reused file. If the modification is not under the permission of the copyright

owner, then such reuse will be at the risk of license violation. License violation in industry can bring large amount of damage for a company. For example, in the legal dispute of Oracle v. Google, Oracle claimed for a penalty $8.8 billion from Google [107]. On the other hand, the open source communities usually take a milder attitude. For example, the Free Software Foundation claims that they do not seek monetary damages when the license is violated [25]. However, as a developer or a company, we should not assume that other developers do not seek for monetary damage and we should always be sure that we do not violate the license. Note that, license inconsistency does not always cause license violation issues, but it is an indication of a bad smell in licensing. The goal of this research is not to eliminate all of the license inconsistencies. Instead, we provide developers with a tool to help review the license inconsistency issues inside their projects to avoid potential legal disputes. We introduce an motivating example of license inconsistency in the next section.

1.1.3 A Motivating Example

In the Debian 7.5 Linux distribution, we observed a file named obstack.c in two packages, `dpkg` and `anubis`. These two files are identical to each other except for their license statements. For this reason we assume that these two files share the same provenance.

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

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

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

```
[...]
This program is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation; either
version 3 of the License, or (at your option) any
later version.
[...]
```

As we can see, the licenses of the two files are different: GPL-2.0+ and GPL-3.0+. The first file (GPL-2.0+) can be combined with software under the GPL-2.0, while the second (GPL-3.0+) cannot (the GPL-2.0 is incompatible with the GPL-3.0). Based on our definition, this is a case of license inconsistency. If this license inconsistency was caused by the reuser

modifying the license without obtaining the permission from the original author, then it indicates potential license violation issues.

We will discuss this issue in details in Chapter 2.

1.2 Software Reuse

Software reuse is an activity based on creating software systems from existing software, rather than building it from scratch [36, 51, 87]. Software reuse has a long history and has been proven to be an effective way to reduce the cost in software development [14, 17, 61]. Jones [46] generalized 4 types of software reuses: *i*) data reuse *ii*) architecture reuse *iii*) design reuse *iv*) code reuse and *v*) module reuse. Our study falls into the category of *code reuse*. Among the sources of code reuse, Q&A platforms serve as an important code base for developers to look for source code and solutions to their programming tasks.

1.2.1 Question and Answer Platforms

A question and answer (Q&A) platform is a type of crowdsourcing, where developers gather and share their questions and solutions to their programming issues. Some popular Q&A platforms include Quora, Yahoo Answers, Stack Overflow etc. Among them, Stack Overflow is the de facto platform for developers while others are for general public. Crowdsourcing is a form of collective intelligence, the general idea being that information processing can emerge from the actions of groups of individuals. In recent years, several collective intelligence approaches have been adopted in software development [54]. Latoza and van der Hoek [54] generalized crowdsourcing into two models: one is *peer production* (e.g., open source, Q&A platforms); the other one is *competition* (e.g., TopCoder). Since the context of this dissertation is code reuse, and the competition model has no clear path of code reuse (i.e., source and destination), we will focus on Q&A platforms.

These platforms provides a large knowledge/code base for developers, which attracts quite a few numbers of studies. Several studies focused on leveraging the knowledge from the Q&A platforms for software engineering tasks such as documentation generation [89, 94, 108], bug fixing [31] and API call extraction [11]. Other studies proposed approaches to enhance code reuse from these Q&A platforms [66–68, 72].

Although these studies provided approaches/tools to enhance the reuse of discussions/code from Q&A platforms, it is not clear how exactly these source code were integrated into real-world projects. In other words, the process of code reuse is not fully studied, thus we aim to investigate how developers reuse source code from Q&A platforms. If we understand the barriers that prevent developers from reusing source code from Q&A plat-

forms efficiently, we would be able to find out solutions to improve the current reuse process. A motivating example is shown in the next section.

1.2.2 A Motivating Example

In the example shown in Listing 1.1 and 1.2, the developer copied three lines of source code in the accepted answer from the Stack Overflow post and renamed the variable from `hr`, `min`, and `sec` to `hours`, `minutes`, and `seconds`, respectively. Developers are making modifications which does not affect the funcionality of the source code, i.e., cosmetic modifications. We want to understand whether developers are suffering from such modifications during the code reuse. Futhermore, we would like to generalize different types of code reuses, and investigate more barriers that the developers are facing. Details of this study will be explained in Chapter 3.

Listing 1.1: Source snippet from the project.[1]

```
1 hours = TimeUnit.MILLISECONDS
2   .toHours(elapsedTimeMilliseconds);
3 minutes = TimeUnit.MILLISECONDS
4   .toMinutes(elapsedTimeMilliseconds
5   - TimeUnit.HOURS.toMillis(hours));
6 seconds = TimeUnit.MILLISECONDS
7   .toSeconds(elapsedTimeMilliseconds
8   - TimeUnit.HOURS.toMillis(hours)
9   - TimeUnit.MINUTES.toMillis(minutes));
```

1.3 Contributions of the Dissertation

The contributions of this dissertation are two-fold: one is regarding the evolution of software license during code reuse; the other one is regarding the evolution of the source code itself during code reuse.

1.3.1 Software License

The first part of our contribution is regarding the license inconsistency issues. License inconsistency issue is a type of software license issues that is not fully investigated in previous studies. The contribution of our work are:

- We describe and categorize different types of license inconsistencies.

[1] https://goo.gl/9ouSz1
[2] https://goo.gl/74oVBu

Listing 1.2: Source snippet from the Stack Overflow answer.[2]

```
1
2  final long hr = TimeUnit.MILLISECONDS.toHours(l);
3  final long min = TimeUnit.MILLISECONDS
4    .toMinutes(l - TimeUnit.HOURS.toMillis(hr));
5  final long sec = TimeUnit.MILLISECONDS
6    .toSeconds(l - TimeUnit.HOURS.toMillis(hr)
7    - TimeUnit.MINUTES.toMillis(min));
```

- Based on existing tools for license identification and clone detection, we have developed a method to detect license inconsistencies. We perform an empirical study with this method using two sets of FOSS projects. This study reveals that license inconsistencies exist. It also proved the feasibility of our method.

- We perform a manual analysis of some license inconsistency cases to understand the reasons behind them. We then summarized these reasons into 4 categories. Among them, two categories indicate license problems and require developers' attention.

1.3.2 Source Code

The second part of our contribution is regarding the source code, which are:

- We found developers reuse source with certain degree of modifications varying from renaming variables to rewriting the whole algorithm, based on an analysis of how developers reuse source code from Q&A platforms to their projects.

- We generalize several barriers that prevent developers reusing source code from Q&A platforms, including: (1) too much code modification required to fit in their projects, (2) incomprehensive code, and (3) low code quality.

- We summarize developers' suggestions for next-generation Q&A platforms along the following dimensions: (1) code quality, (2) information enhancement & management, (3) data organization, (4) license, and (5) human factor.

1.4 Overview of the Dissertation

The rest of this dissertation is organized as follows:

Chapter 2 reports our work on analyzing license inconsistency and its evolution based on an empirical study of Debian 7.5 and a collection of Java projects.

Chapter 3 reports our work on investigating the barriers of code reuse. Based on an exploratory study on how developers reuse source code from Q&A platforms, we performed a survey on 453 developers. This study gives us insights of how to create a next-generation Q&A platforms with the expectation of improve the code reuse for developers.

Chapter 4 concludes this dissertation and shows directions for future work.

Chapter 2

Analysis of License Inconsistency in Large Collections of Open Source Projects

2.1 Introduction

Software reuse has long been advocated as a good practice to reduce development time and increase product quality [14, 17, 36, 51, 61, 87]. The popularity of Free and Open Source Software (FOSS) has made software reuse a common practice. FOSS software can be defined as software that is licensed under a free or open source license. In a nutshell, a free and open source license allows the software to be freely used (as in freedom), modified, and redistributed (in modified or unmodified form) by anyone, as long as the conditions of its license are satisfied. The Open Source Initiative (OSI) has defined a set of characteristics that an open source license should have, and published a list of approved Open Source licenses[1]. The Free Software Foundation[2] defines a set of similar conditions that a license should satisfy in order to be considered a free software license.

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

[1] `http://opensource.org`
[2] `http://www.fsf.org`

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

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

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

Previous study by Li *et al.* [55] shows that 36% of the developers who reused the OSS components changed the source code, but they did not point out whether these changes involve the license statement. In our study, we focus on the license statement changes and the license inconsistency introduced between the different copies of the files.

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

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

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

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

The contributions of this work are:

1) We describe and categorize different types of license inconsistencies.

2) Based on existing tools for license identification and clone detection, we have developed a method to detect license inconsistencies. We perform an empirical study with this method using two sets of FOSS projects. This study reveals that license inconsistencies exist. It also proved the feasibility of our method.

3) We perform a manual analysis of some license inconsistency cases to understand the reasons behind them. We then summarized these reasons into 4 categories. Among them, two categories indicate license problems and require developers' attention.

This chapter is organized as follows. Section 2.2 describes background on FOSS licenses and license inconsistencies. Section 2.3 introduces our research method. Our empirical study that uses this method is described in Section 2.4, followed by Section 2.5 with a discussion of the results. Section 2.6 describes threats to validity. After a description of related work in Section 2.7, Section 2.8 concludes this chapter and points out the future direction.

2.2 License Inconsistencies

A software license is a permission to reproduce, modify and redistribute a software, usually granted under certain conditions. An open source license is a software license that follows Open Source Definition[4] and is approved by the Open Source Initiative. As of today, only 82 licenses have been approved as Open Source License [5]. However Black Duck Software claims that the Black Duck Knowledge Base includes over 2200 licenses[6]. Some licenses have been grouped under the same name as different versions. For example, the General Public License (GPL) has versions 1, 2 and 3. Each version is, in legal terms, a totally independent license.

To reuse OSS source code files, developers must identify the license under which the files are made available, understand their terms, and satisfy their requirements. This is not a trivial task because some open source licenses do not usually allow easy integration with software under another license (see German and Hassan [34] for a detailed discussion on this issue).

[4] http://opensource.org/definition
[5] https://opensource.org/licenses/alphabetical
[6] http://www.blackducksoftware.com/products/knowledgebase

For example, software under the Apache Public License version 2 (Apache-2.0)[7] can be reused and integrated into software licensed under the GPL-3.0. On the other hand, software under the GPL-2.0 cannot be combined with software under the GPL-3.0 (however software under the GPL-2.0+, that is version 2 or any later version of the GPL, can be). Therefore, developers must know the licenses of files they reuse in order to avoid license violations.

It is also known that frequently, the source code files in an application are under different licenses [59, 60]. In addition, copies of the same file might have different licenses because the copyright owner has licensed the file accordingly. For example, the copyright owner has decided to change the license from one version of the software to the other (even if the software did not have any changes).

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

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

```
/* Getopt for GNU.
* NOTE: getopt is part of the C library, so if you don't
* know what "Keep this file name-space clean" means, talk
* to drepper@gnu.org before changing it!
* Copyright (C) 1987-1996, 1998-2004, 2006, 2008-2012 Free
* Software Foundation, Inc.
* This file is part of the GNU C Library.
*
* This program is free software: you can redistribute it
* and/or modify it under the terms of the GNU General Public
* License as published by the Free Software Foundation;
* either version 3 of the License, or (at your option) any
* later version.
*
* This program is distributed in the hope that it will be
* useful, but WITHOUT ANY WARRANTY; without even the implied
* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
* PURPOSE. See the GNU General Public License for more
* details.
```

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

```
*
* 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 statement of a source file can only be modified by its copyright owner. Reusers shall never modify the license statement unless it is under the permission of the copyright owner or allowed by the terms of the license.[8] Otherwise, the reusers may incur a license violation.

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

2.2.1 Definition

For the purpose of this research, a *license inconsistency* refers to the situation where two source files contain the same source code but have different license statements.

2.2.2 Example

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

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

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

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

```
[...]
This program is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation; either
```

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

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

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

1) The first file is the original one and was copied to the second project, where the license was changed from GPL-2.0+ to GPL-3.0+. In this case, because the original license allows to use newer versions of the license, the change can be done by anybody, and it is not a potential license violation.

2) The second file is the original one and was copied to the first project. The license version was changed from GPL-3.0+ to GPL-2.0+ in the first project. This could be a potential violation if the change was made without the approval of the copyright owner of the file.

3) Both of the files are copied from the same third-party project (who created the file). Each project made the copy at different times, one before, and one after the license of the file was changed by the original project copyright owner. In this case, there is no potential license violation.

To determine which one is the actual reason of the inconsistency, we need to examine the repository history of these two projects and try to determine the true origin and if possible, identify the rational for this modification of license. This topic will be discussed in Section 2.4.1.

2.2.3 Categorization

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

1) **License Addition**: The source file was without a license, and a license is added in a later time.

2) **License Removal**: The source file was under a certain license, and the license is removed in a later time.

3) **License Upgrade**: The source file was under a certain version of the GPL license—a license that allows an upgrade (such as the GPL-2.0+ and GPL-3.0+)—and it is upgraded to a newer version of the license.

4) **License Downgrade**: The source file was under a certain version of a license, and it is downgraded to an older version of the same license.

5) **License Change**: The source file was under a certain license, and it is changed to another license (except for License Upgrade and License Downgrade).

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

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

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

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

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

2.3 Method to Detect License Inconsistencies

In our previous work [110], we have proposed a method that can efficiently detect license inconsistencies. However, a major issue with that method is

that it only considers license inconsistencies among files that have the same file name (in order to achieve a fast performance). Thus if files are renamed during the process of copy-and-own reuse, a license inconsistency will not be detected. To solve this problem and make our result cover more license inconsistency cases, we propose a new method in this chapter. A detailed comparison of these two methods will be discussed in Section 2.5.1.

In our new approach, we focus on detecting license inconsistencies among file clones. In the scenario of source code reuse where source files are imported from an upstream project, the contents of reused source files remain almost the same, sometimes with small changes (such as modifying comments, renaming identifiers etc.) [74].

To decide whether source files are copies of each other—or in other words whether they share the same provenance—we compare their *normalized token sequences* [73]. Normalized token sequences are generated from the source file by removing the comments, redundant white spaces, new lines, carriage returns and then converting identifiers to normalized tokens. If two files have the same normalized token sequences, then it is likely that they are copies of each other and we call them *file clones*, which are actually Type-2 code clones [73, 74]. We use `CCFinder` [48], a code clone detection tool, to analyze and determine if files are file clones. `CCFinder` will generate a preprocess file which contains the normalized token sequences of the source file. For those file clones with the same normalized token sequences, we assume that they come from the same origin, and then gather them into the same *file group*. Files in the same file group might have different file names but similar program statements, possibly with different comments including license statement.

Once that we group these similar files, we identify the license of the files in each group. In our approach we used `Ninka` to detect the license of source files, since `Ninka` is reported to have the highest precision of all the license detection tools including `FOSSology`, `ohcount` and `OSLC` in the research by German *et al.* [35]. `Ninka` is a sentence-based license detection tool which can identify 110 different licenses with 93% accuracy, and it can handle more than 600 files per minute. There are two special results from `Ninka`: one is *UNKNOWN*, which represents that `Ninka` has found a license but does not recognize it. The other one is *None*, which states that the source file has no license.

We then compare the licenses of each file in the license list of each group. If all the files have no license, or all of them have the same license, then there is no license inconsistency. Otherwise, the group is likely to contain one or more types of license inconsistencies. And then, based on the relation between licenses, our approach identifies the type of license inconsistency. Note that a group may have multiple types of license inconsistencies. For example, if a group consists of a file under GPL-2.0+, a file under GPL-3.0+

Table 2.1: Strategies to decide whether a certain type of license inconsistency exists in a group.

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

and another file under Apache-2.0, then the group has two types of license inconsistencies: *LUD* between GPL-2.0+ and GPL-3.0+, *LC* between GPL-2.0+/GPL-3.0+ and Apache-2.0. For such reason, we calculate *License Inconsistency Metrics* for each of these groups, from which we can measure what type of license inconsistency and how many of each type exist in the groups.

2.3.1 License Inconsistency Metrics

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

#**File:** Number of files in this group.

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

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

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

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

These metrics are calculated for each file group based on their license lists. The strategies shown in Table 2.1 enable us to decide whether a certain type of license inconsistency exists in this group.

Specifically, if we query the metrics result for each group based on the conditions of $\#None > 0$ and $\#Lic > 0$, which mean respectively that

there is one or more files with no license(s), and that there is one or more files with a license, we get what we define as ***LAR*** (a license addition or removal); if we query for those whose $\#GPL \geq 2$, a condition which means that there are two or more different licenses in the GPL family (such as GPL-2.0+ and GPL-3.0+), we get ***LUD*** (a license upgrade or downgrade); and if we query for those based on $\#GPL \leq 1$ and $\#Lic \geq 2$, which mean respectively that there are more than two licenses in this group and that there is no more than one GPL license (excluding ***LUD*** cases), we get ***LC*** (a license change) where one license is changed to another one.

2.3.2 Method of Detecting License Inconsistencies

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

1. **Create groups of file clones:** For all the source files in the target projects, we apply `CCFinder` to extract the normalized token sequences of each file. Note that, although `CCFinder` itself is a clone detection tool, we do not utilize the full functionality of `CCFinder` and we only use it to generate the normalized token sequences of source files. By computing and categorizing the hash value of these token sequences, we then create a *group* for files that have the same normalized token sequences. Each group contains at least two different files; i.e., a unique file is not contained in any group.

2. **Identify licenses for files in each group:** For each group of file clones, `Ninka` is used to identify the license(s) of each file. The result is a list of licenses for each file group.

3. **Report groups that contain a license inconsistency and calculate the inconsistency metrics:** We compare the license list of each file group. File groups are reported to have license inconsistencies unless all the licenses on the list are exactly the same. The result is a list of file groups that contain one or more types of license inconsistencies.

2.3.3 Example

We illustrate our method with a project shown in Figure 2.1. This project consists of 4 packages. The source code of foo.c file in `Pkg2` is exactly the same with the one in `Pkg1`, but the license statement is changed from GPL-2.0+ to GPL-3.0+; The source code of foo.c in `Pkg3` is different from the one in `Pkg1`, i.e. they happen to have the same file name. It is reused in `Pkg4` with its name changed to foo100.c and license statement removed.

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

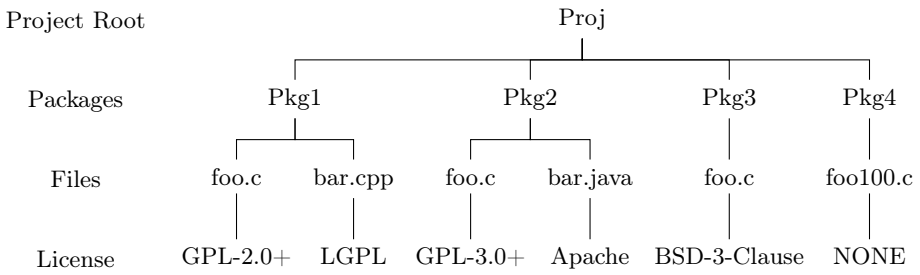

Figure 2.1: Hierarchy of a project and the license of each source file. Note that the foo.c file in `Pkg1` was imported to `Pkg2` with the license changed to GPL-3.0+; The foo.c in `Pkg3` contains totally different source code than the one in `Pkg1`, and was imported to `Pkg4` with its name changed to foo100.c and license removed.

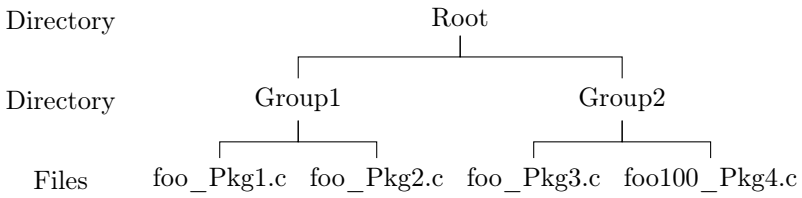

Figure 2.2: Hierarchy of the grouped files.

and `Pkg2` have the same source code (except for their code comments which include license statement), `CCFinder` treats them the same, and generate the same token file. This also applies to foo.c file from `Pkg3` and foo100.c from `Pkg4`. Thus we can compare the hash value of the token files and group them into two groups, as shown in Figure 2.2.

2. **Identify licenses for files in each group:** For each file in the group, we use `Ninka` to detect their licenses and make a list of the file name, group index and the licenses, as shown in Table 2.2. *File name* is the name of the source file. *GroupID* indicates the index we use to identify file groups.

3. **Report groups that contain license inconsistencies and calculate inconsistency metrics:** We examine the licenses of each group and found that both of these groups contain license inconsistencies.

Thus we report both of these groups and compute the inconsistency metrics for each of them, as shown in Table 2.3.

Table 2.2: License list of the selected files from the example project.

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

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

2.4 Empirical Study

We have selected two target datasets for analysis: Debian 7.5 Linux distribution[9] and 10,514 Java projects randomly downloaded from GitHub[10]. We then conducted our method on both datasets respectively. Since it is hardly feasible to determine how many and what types of license inconsistencies are there in the target projects, it is difficult to get an oracle data set and to perform a quantitative evaluation of our method, specially regarding its recall, which we will talked more about in Section 2.5.4. However, a qualitative evaluation of this method is discussed in Section 2.5.

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

[9] https://www.debian.org/
[10] https://github.com/

Table 2.3: List of the license inconsistency metrics for each file group in the example project.

GroupID	#File	#Lic	#None	#Unknown	#GPL
1	2	2	0	0	**2**
2	2	**1**	**1**	0	0

Table 2.4: Main characteristics of Debian 7.5.

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

2.4.1 Empirical Study on Debian 7.5

We conducted our study using a large open source Linux distribution, Debian 7.5. The source code was downloaded from its official site and its main characteristics are shown in Table 2.4. Only .cpp, .c and .java files are used, since they account for the majority of source code in the Debian distributions and are the file formats supported by `CCFinder`.

Results

In the first step, we grouped the files under each set by their normalized token sequences and resulted in 125,092 groups in total. The number of files within one group ranges from 2 to 160, and the average number of files per group is 2.8 with a median value of 2. The breakdown of each file type is shown in Table 2.5.

Table 2.5: Breakdown of number of groups and files for each type in analyzing Debian 7.5.

File type	#Group	#Files	#Mean	#Median
.c	68,568	207,620	3.0	2
.cpp	16,202	38,617	2.4	2
.java	40,322	108,868	2.7	2
Total	125,092	355,105	2.8	2

Completing the following two steps, 6,763 groups were reported to have at least one type of license inconsistency, which is 5.4% of the 125,092 groups in total. For the sake of space, we show only three of them in Table 2.6, representing each of the three types of license inconsistencies, which will be discussed in the rest of this section.

Then we calculate the number of each type of license inconsistency and their proportion. The result is shown in Table 2.7. From this table, we can see that from the total of 6,763 groups that contain one or more

Table 2.6: Partial list of the license inconsistency metrics for each file group in detecting Debian 7.5.

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

* Unknown licenses.

** Each group may contain files with different file names. In this case we choose the majority file name to represent that group.

*** These two groups both contain files named getopt.c, but the source code between these two groups are totally different.

Table 2.7: Number of different types of license inconsistencies and their proportion in Debian 7.5. Note that one group may contain more than one inconsistency types, so that the total percentage can exceed 100%.

Inconsistency type	Frequency	Perc.
LC	4,562	67.5%
LUD	2,137	31.6%
LAR	883	13.1%

license inconsistency cases, 67.5% of them contain *LC*, followed by *LUD* and then *LAR*. Further study is needed to investigate the legality of these modifications.

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

– ***LAR*:**

Examining the getopt.c in the second line from the inconsistency result list in Table 2.6, we get the license list of that group in Table 2.8. The remaining files that contain the same licenses are omitted from this list.

Table 2.8: Example of *LAR* inconsistency, in getopt.c

Package name	License
icedove	NONE
iceweasel	MPL-2.0

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

getopt.c from `icedove` package:

```
#include <stdio.h>
#include <string.h>
[...]
int main(int argc, char **argv)
{
    PLOptState *opt;
    PLOptStatus ostat;
    [...]
    return 0;
}
```

getopt.c from `iceweasel` package:

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

As we can see in the file from `icedove` package, there is no license statement at all, while the file getopt.c from `iceweasel` package contains a MPL-2.0 license. Meanwhile, the other parts of these two files are exactly the same, hence we consider it safe to assume that the origin of both files is the same. There are several possible explanations to this case of license inconsistency:

1. The file from `icedove` package is the original one, and the developers of `iceweasel` project reused the file and added a license to it.

2. The file from `iceweasel` package is the original, and developers of `icedove` project reused this file and removed the license statement.

3. Both of the files in these two projects reused different versions of this file from another project (where the license was added or removed).

Table 2.9: License list of group 6645 of obstack.c where *LUD* exists.

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

Table 2.10: License list of group 52662 of getopt.c where *LC* and *LAR* exist.

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

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

– ***LUD*:**

To exemplify this type of license inconsistency, we will use obstack.c, which is in the first line in Table 2.6. Table 2.9 shows two packages that reuse this file. As we can see from this table, the first file is licensed under GPL-2.0+ while the second one is under GPL-3.0+.

The license statements of the files from `dpkg` and `anubis` package were listed in Section 2.2.2. Both of these files contain more than 400 lines of code, and they are exactly the same except for their license statements. Tracing the file history in both projects we found that this file was originally created in `gnulib`. The license of this file was upgraded in `gnulib` from GPL-2.0+ to GPL-3.0+. By examining the commit log of `dpkg`, we found that the developers of `dpkg` intentionally reused the older version of the file from `gnulib` project (they wanted the file to be licensed GPL-2.0+, not GPL-3.0+), which caused the license inconsistency.

– ***LC*:**

We demonstrate this type of license inconsistency using getopt.c in the third line from the Table 2.6.

As shown in Table 2.10, getopt.c from snort package contains GPL-2.0 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
[...]
```

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

Manual Analysis

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

We randomly chose the samples. To be precise, first we randomly selected a case of license inconsistency and investigated the reason that this case occurs, then we randomly selected the next case and repeated the process. The time needed to investigate each case varies from several minutes to several hours, depending on how well the related project is documented. Due to the difficulties and the time invested, we stopped investigating cases when the reasons are saturated, that is, when same reasons of license inconsistency kept coming up as we investigate new cases. Based on this policy, we have investigated 25 cases in total. Then we tried to categorize them according to the reason that caused such inconsistencies. They are divided into three categories: safe changes (no violation is found), unsafe changes (given all information available, it appears to be a violation) and uncertain (it was not possible to determine whether it was safe or unsafe).

Table 2.11: The count and percentage of each category for the 25 investigated license inconsistency cases.

Category	#	**Perc.**	**Sub-category**	#	**Perc.**
Safe changes	14	56%	Original author changed the license.	10	40%
			Reuser chose a license from a multi-license.	4	16%
Unsafe changes	6	24%	Reuser changed the license.	5	20%
			Reuser added one or more licenses.	1	4%
Uncertain cases	5	20%	Source files are too small to be considered as clones.	2	8%
			Source files cannot be found in the upstream repositories.	1	4%
			Repositories are not available.	2	8%
Total	25			25	100%

The results are shown in Table 2.11, and the a detailed explanation of each category is as follows:

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

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

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

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

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

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

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

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

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

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

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

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

From the result we examined a file named DOMException.java. This author of this file is World Wide Web Consortium (W3C), and was licensed under W3C Software License. When developers reused this source file in `ikvm` project, they added a GPL-2.0 License to it resulting a composition

of these two licenses. Meanwhile, the program code of this file was not changed at all. We consider this case as unsafe, since this type of license modification makes it unclear which part contains the original license and which part contains the new license, since they added the license without adding any source code changes to the file.

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

1) Source files are too small. Some files contain the same source code, but due to their small size (e.g., less than 10 lines of code) it is difficult to decide whether one is reused by the other or they just happen to be the same. A more detailed case is discussed in Section 2.5.2.

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

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

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

One example is, when we tried to checkout the source code of `axis` project using the SVN command found on its official website[11], the command returned an error that the URL does not exist.

2.4.2 Empirical Study on Java Projects

The other data set we studied is a collection of 10,514 Java projects randomly cloned from GitHub. The snapshot was taken in Mar. 2015, and only those projects that consist of at least 100 commits are selected. Table 2.12 shows the characteristics of these projects. Since .java files are 98.9% of all the files, we will focus our following analysis on them only.

Results

In the first step, source files are grouped by their normalized token sequences. The result was 199,284 groups. The number of files within each

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

Table 2.12: Main characteristics of Java projects cloned from GitHub.

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

Table 2.13: Number of groups and files in each group in analyzing Java projects.

File type	Group count	#Files	#Mean	#Median
.java	199,284	769,220	3.9	2

group ranges from 2 to 1514, and the average number is 3.9 with a median value of 2, as shown in Table 2.13.

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

Furthermore, the number and proportion of each type of license inconsistency is shown in Table 2.14.

Manual Analysis

As we did in the Debian study, we examined a random sample of the inconsistent groups. We sampled 17 cases, and tried to categorize them according to the reason that caused such inconsistencies. As described before, they are divided into three categories, the percentage of each category is shown in Table 2.15, and the explanation to each category is as follows:

– **Safe Changes:**

Table 2.14: Number of different types of license inconsistencies and their proportion in Java projects.

Inconsistency type	Number	Perc.
LC	12,653	90.9%
LAR	6,179	44.4%
LUD	1,316	9.5%

Table 2.15: The count and percentage of each category for the 17 investigated license inconsistency cases in the Java projects.

Category	#	**Perc.**	**Sub-category**	#	**Perc.**
Safe changes	11	65%	Source files are in the same project but with different licenses.	8	47%
			Duplicated projects are not up-to-date.	2	12%
			Reuser added a same license to the source file.	1	6%
Unsafe changes	1	6%	Reuser modified the license terms.	1	6%
Uncertain cases	5	29%	Licenses are modified outside the scope of their repositories.	1	6%
			Source files are too small.	4	24%
Total	17			17	100%

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

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

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

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

We examined two projects: `JCrypTool`[12] and `JCT-CA`[13]. A file named ResizeHelper.java exists in both projects with the same normalized token se-

[12] `https://github.com/jcryptool/crypto`
[13] `https://github.com/Kalliope/minica`

quences. The one in JCT-CA is without a license, while the one in JCrypTool was originally with no license but then added with a EPL-1.0. The readme file from JCT-CA states:

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

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

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

– **Unsafe Changes:**

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

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

– **Uncertain Cases:**

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

2) Source files are too small. This case is same as the one in Debian data set. This issue will be discussed in Section 2.6.

For example, a file named ReaderInputStream.java was found in bingo-core project with an Apache-2.0 license and in hibernate-orm project with an

LGPL license. However, the source code contents of these files are quite small, which merely contains two empty constructor methods. The source code part excluding the comments is shown as following:

```
[...]
import java.io.IOException;
import java.io.InputStream;
import java.io.Reader;
public class ReaderInputStream extends InputStream {
        private final Reader reader;
        public ReaderInputStream(Reader reader){
                this.reader = reader;
        }
        @Override
        public int read() throws IOException {
                return reader.read();
        }
}
```

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

2.4.3 Discussion of the Results

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

The manual analysis of several cases of license inconsistencies gives us a rough understanding of how many of these cases are safe or not. From Table 2.11 and Table 2.15 we can see that, both in Debian 7.5 and Java projects we selected, unsafe and uncertain cases take up 44% and 35% respectively. This shows that it is not uncommon that license inconsistencies might lead to potential license violation problems.

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

Many files in an open source project are frequently imported from other projects. It is not a trivial task to find the repositories of these upstream projects. Take the Debian distribution as an example: some of the packages contain a file indicating the repository URL of that package, but some do not. For such packages, we needed to search for the official site of the upstream project and try to find its repository URL. There are packages that appear not to use version control systems. They simply provide source

code tarballs for each version on their server. In this case, we have to download each tarball and track the license change manually. This makes provenance tracing more difficult.

In some cases the change of the license statement is not recorded in the revision history because the license statement is changed (we presume) before the file is added to the repository's project. In this case, we have to check other information (e.g. on the official site of the project or in the commit comment where the file was added) to find out the reason why developers changed the license. Our results are consistent with Vendome *et al.* [95], who found a lack of traceability for license changes.

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

2.4.4 Answering RQs

Revisiting the research questions:

- **RQ1:** *How can we categorize a license inconsistency?* We categorize license inconsistencies into these 3 types: *i*) `LAR`, which is typically caused by license addition or removal; *ii*) `LUD`, which is related to license upgrade or downgrade in the GPL family; *iii*) `LC`, which is usually caused by license change in the process of license evolution.

- **RQ2:** *Do license inconsistencies exist in open source projects?* Yes, license inconsistencies exist in open source projects. As we have shown in our empirical studies of Debian 7.5 and a large collection of Java projects on GitHub, various types of license inconsistencies were detected.

- **RQ3:** *What is the proportion of each type of license inconsistency?* In the case study of Debian 7.5, out of 125,092 file groups, 5.4% of them contain one or more license inconsistency cases. The proportion of each type is: `LAR` (13.1%), `LUD` (31.6%) and `LC` (67.5%). In the case study of Java projects, out of 199,284 file groups we selected,

7.0% of them contain one or more license inconsistency cases. The proportion of each type is: `LAR` (9.5%), `LUD` (44.4%) and `LC` (90.9%).

- **RQ4:** *What caused license inconsistencies? Are they legally safe?* The reasons that caused license inconsistencies can be summarized into these groups according to our observation:

  i) Original author modified/upgraded the license.

  ii) The file was originally multi-licensed and reusers chose either one.

  iii) Reuser added one or more licenses.

  iv) Reuser appears to have replaced the original license, and changed the copyright owner.

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

2.5 Discussion

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

2.5.1 Improvement of the Method

As described in Section 2.3, our previous method [110] omits the cases if the files are renamed during the process of copy-and-paste reuse to achieve higher performance.

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

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

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

Debian 7.5

Table 2.16 shows the comparison of results obtained by the two methods, for Debian 7.5.

Table 2.16: Comparison of two methods on Debian 7.5.

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

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

Table 2.17: Comparison of two methods on Java projects.

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

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

Java Projects

Table 2.17 shows the comparison of results obtained by the two methods when applied to the Java projects in GitHub.

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

2.5.2 What Appears to Be a Copy Might Not Be a Copy

We sent emails to the 3 development teams of the projects where unsafe license modification were found, to understand why they modified the license and whether they consider it as an illegal modification and two of them replied us. One of them claimed that they wrote the source code all

from scratch, and denied that this source file was copied from somewhere else. This source file was so small which contains merely two empty constructors, thus we believe it is possible that different developers happen to create the same file. Note that, this is not a false positive case of our method, since our method is designed to detect *license inconsistency* cases in the target projects, not the *license violation* cases. However, it stresses the need to consider a minimum size threshold, in order for these small files not be considered in the analysis.

2.5.3 Changes Were Made Under the Permission of Copyright Owner

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

2.5.4 An Attempt in Measuring the Recall

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

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

2.6 Threats to Validity

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

Meanwhile, source code files are evolving: those that come from the same provenance may differ from each other dramatically after being modified by developers, resulting in different normalized token sequences thus making our method fail to detect these license inconsistency groups. A possible solution to this problem might be using pairwise checking method instead, e.g. detect clones based on the similarity of each pair of source files. However, in a conventional pairwise checking method, n^2 pairs of source files need to be processed when there are n source files in total. In our approach, we only need to calculate the hash value of the normalized token sequences of each file, and the files that have the same hash value would be naturally put into the same clone group. Thus the time complexity of our approach would be $O(n)$ instead of $O(n^2)$ in the pairwise checking method. Due to this performance reason, we chose to use hash approach instead of pairwise checking method in this chapter. And since we can still get large numbers of file groups that contain license inconsistencies using this method, we believe that it is good enough for this exploratory study.

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

One aspect that is important to highlight is that our method relies on the ability to detect copies of files. In our previous paper [110], we found copies of files by analyzing files with the same name. In this study we compared the normalized token sequences of files. We could also do full clone detection and consider two files to be copies of each other only if they were above certain threshold. This process would have been significantly more time consuming. Ultimately, detecting license inconsistencies is a balance between performance of the detection vs. recall. If necessary, step one of our method can be replaced with other methods that provide better recall, at the expense of being slower, and potentially require more manual analysis to filter out false positives.

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

In the process of license identification, as we used `Ninka` to identify the license of source files, its accuracy should also be considered. German et al. reported that the accuracy of `Ninka` is 93% [35]. We believe this is sufficiently high, so that the license detection result is good enough to support our analysis. In addition, we regard UNKNOWN licenses as the same license within each group, different from any other licenses. If these UNKNOWN licenses in a same group are actually different from each other, we may underestimate the number of license inconsistency cases. But this concern is mitigated according to our observation to these UNKNOWN licenses: most of those in the same group actually contain the same license statement, either a license that is not approved by OSI or a user modified version of an OSI-approved license. On the other hand, if these UNKNOWN licenses are actually the same as those recognized ones (e.g. GPL-2.0, BSD-3-Clause etc.) in the same group, this could be considered as a false positive. In this case, these UNKNOWN licenses are not exactly the same as the original license, meaning that someone must have modified the license statement (making `Ninka` not able to recognize the license). We believe that it is necessary to check whether these changes are legal or not. Thus it is reasonable to treat them as license modifications, which is consistent with our assumption. To obtain more precise results, it is necessary to improve the accuracy of license identification.

2.7 Related Work

Many studies address inconsistent changes among code clones. Krinke [50] studied on changes applied to code clones in open source software systems and showed that half of the changes to code clone groups are inconsistent changes and these changes are not solved if they occurred in a near version. Göde and Harder [39] studied patterns of consecutive changes to code clone in real software systems. Some approach to find inconsistent changes are proposed [27, 43]. On the other hand, Bettenburg *et al.* [15] showed that only 1% $\sim$ 4% of inconsistent changes to code clone introduce software defects. In addition, Göde and Koschke [40] 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 [35, 38, 92]. Using these approaches, some researches analyzed software licenses in open source projects and revealed some license issues. Di Penta *et al.* [23] 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.* [33] proposed a method to understand licensing compatibility issues in software packages. They mainly focused on the compatibility between license declared in packages and those in source files. In another research by German *et al.* [32], 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.* [5] proposed an approach for calculating conflicts between licenses in terms of their conditions. However, our work proposed an approach to find license inconsistencies in similar files. By investigating the revision history of these files, we summarized the factors that caused these license inconsistency cases and tried to decide whether they are legally safe or not. Zhang *et al.* [118] proposed an automatic method to check license compliance problems caused by ignorance or carelessness. They use Google Code Search to discover file clones with different licenses, while our method detect file clones within our target data sets. Recently Vendome *et al.* [95] performed a large empirical study of Java applications and found that changing license is a common event and a lack of traceability between when and why the license of a system changes. In their following research, Vendome *et al.* [96] investigated the reasons on when and why developers adopt and change licenses during evolution of FOSS Java projects on GitHub by conducting a survey with the relevant developers. They concluded that developers consider licensing as an important task in software development. However, license implications or compatibility are not always clear and so they can lead to changes. In addition, other external factors, such as community, purpose of usage and use of third-party libraries also influence the projects' licensing.

2.8 Conclusion of This Chapter

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

show the existence of license inconsistencies in open source projects and prove the feasibility of our method.

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

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

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

For future work, we will apply our tool to more projects and examine the proportion of each type of license inconsistency. We are especially interested in applying this method to industrial projects and see whether closed source projects comply with the license terms of open source licenses. With the increased number of projects, we believe that many more license inconsistency cases will be found. And we will try to make a quantitative evaluation of this tool. Furthermore, we will try to develop a method to help us analyze the history of each file, so that we can decide the safety of these inconsistencies efficiently.

Chapter 3

How Do Developers Utilize Source Code from Stack Overflow?

3.1 Introduction

Technical question and answer (Q&A) platforms such as Stack Overflow have become more and more important for software developers to share knowledge. Developers can post questions on these Q&A platforms, which in turn are answered by other developers. These answers often contain source code snippets. As of August 2017, Stack Exchange reports that there are approximately 7.6 million users and 14 million questions with 23 million answers on Stack Overflow [83]. Among those answers, 15 million (75%) have at least one source code snippet attached, which forms a huge code base for developers to reuse source code from.

However, reusing source code is not easy [29]. For example, these are two of the challenges that developers face when reusing source code from Q&A platforms:

(1) It is difficult for developers to find suitable source code based on their particular needs, such as language, functionality, and performance [98]. To address this challenge, a number of studies have been done to help developers to locate more relevant source code snippets [67, 68, 98, 103].

(2) Even if developers are able to find suitable source code, it may be difficult to integrate the code in their own projects. For example, parameters may need to be adjusted, or additional source code may need to be added [22]. To address the challenge of code integration, various techniques have been proposed [22, 44, 63, 64, 114], such as automatically renaming variables to make the code fit in the required context.

Prior studies [13, 36, 90, 112] on reusing source code from Q&A platforms have mostly focused on helping developers to locate relevant source code, and on integrating that source code into their own project. In our study, however, we build upon these studies by investigating *how* developers utilize code from Q&A platforms. Such knowledge will help us better understand the potential barriers that developers face when reusing code from Q&A platforms. In this chapter, we use this knowledge to provide a roadmap for improving source code reuse on next-generation Q&A platforms.

We first conduct an exploratory study of 289 source code files from 182 open-source projects, which contain at least one link to a Stack Overflow post. We manually study each file and its linked Stack Overflow post, to investigate how developers reuse code from Stack Overflow. We found that:

- In 44% of the studied files, source code had to be modified before it could be used in the developer's own project. The required modification varied from simple refactorings to a complete reimplementation. This finding provides empirical evidence for the importance of prior studies on automatic code integration [4, 24, 105].

- In 12.5% of the studied files, developers reimplemented code based on the idea of a Stack Overflow answer, which suggests that Q&A platforms should consider to summarize key points that are discussed in a post to give developers a quick overview of a question and its answers.

- Developers reuse source code from non-accepted answers (26%) for several reasons, such as the simplicity and performance of the source code. Some developers even adopt answers that are total opposites from what the original asker wanted but meet their needs. Hence, Q&A platforms should consider to improve the way of organizing answers, so that developers can find the most suitable answers based on their requirements easily, such as voting on the different aspects (e.g., readability or performance) of answers or adding tags for answers.

To further understand the barriers that developers face when reusing code and to collect suggestions for improving the code reuse process on Q&A platforms, we conducted a survey of 453 open-source developers who are also on Stack Overflow. We highlight our findings as follows:

- **Slightly more participants prefer reimplementing source code over reusing source code from Q&A platforms. The reasons are the difficulty of having to make the code fit in their own projects, and a low comprehension or low quality of the code from Q&A platforms.** These findings provide empirical evidence

for the importance of research on code integration and code comprehension, and highlight the need of providing code quality indicators on next-generation Q&A platforms.

- **80% of the participants do not have a good understanding of the licenses of the Q&A platforms. In addition, 57% of the participants think that having more information about the code license is important.** These findings suggest that next-generation Q&A platforms should make code licensing information more visible to developers.
- **The most popular suggestion category for improving code reuse on next-generation Q&A platforms was code quality (35% of the suggestions).** We categorized the suggestions for next-generation Q&A platforms into five categories: code quality, information enhancement & management, data organization, license, and human factor. A large part of the code quality suggestions were about adding an online code validator (42.2%) and a detection mechanism for outdated source code (29.7%).

In summary, the difficulty of fitting code in their own projects, a low comprehension and a low quality of the code are the top barriers that prevent developers from reusing code. Lacking a good understanding of code license is also an important barrier for code reuse. Thus, future studies are encouraged to address these barriers to better facilitate code reuse for developers.

The rest of this chapter is organized as follows. Section 3.2 introduces the background and related work of our study. Section 3.3 introduces our research questions and describes our data collection process. Our exploratory study is described in Section 3.4. Section 3.5 presents the survey design. Section 3.6 summarizes and analyzes the survey results, and presents our roadmap for next-generation Q&A platforms. Section 3.7 describes the threats to validity. And finally, Section 3.8 concludes the chapter.

3.2 Background & Related Work

In this section, we give background information and discuss related work about one of the most popular technical Q&A platforms, Stack Overflow. We discuss Stack Overflow along four dimensions: leveraging knowledge, understanding the quality of posts, source code reuse, and code licensing.

3.2.1 Leveraging Knowledge from Stack Overflow

Nowadays, technical Q&A platforms, with Stack Overflow being the most prominent, have become an important way for researchers and practitioners

to obtain knowledge about and find solutions to their programming problems [1, 91, 104]. Developers are allowed to post questions, answer questions and vote on questions or answers on Q&A platforms. When posting questions or answers, developers often attach snippets of source code to explain their questions or answers along with the textual description. For example, Figure 3.1 shows an example of an answer that contains source code on Stack Overflow. The asker asked how to get the HTML of a selected object with the jQuery library, and the answerer posted an answer that provides a solution in the form of the attached source code.

During the process of asking and answering questions, Stack Overflow accumulates a large amount of knowledge. To leverage the vast amount of knowledge on Stack Overflow, several approaches have been proposed. Treude and Robillard [89] presented a machine learning based approach, SISE, to augment API documentation using answers on Stack Overflow. Gao *et al.* [31] proposed an automated approach to fix recurring crash bugs by leveraging information (e.g., questions with similar crash traces) on Stack Overflow. Azad *et al.* [11] proposed an approach to extract API call rules from version history and Stack Overflow posts. Chen *et al.* [20] proposed an automatic approach to build a thesaurus that contains morphological forms of software engineering terms. These studies make use of the text and code information on Stack Overflow to automatically generate or enrich existing software artifacts and show promising results. Our study is different from these studies as we are interested in studying *how* developers utilize knowledge (i.e., source code) from Stack Overflow.

3.2.2 Understanding the Quality of Posts on Stack Overflow

Stack Overflow allows askers to mark at most one answer as the "accepted answer" to indicate whether the answer meets their requirement (see Figure 3.1). Stack Overflow allows developers to upvote or downvote a post (e.g., a question or an answer) to express whether the post is useful. The total number of up and downvotes that a post receives is displayed as a score next to the post. For example, the score of the question in Figure 3.1 is 673. In general, answers and questions with a high score are usually regarded as high-quality ones.

Several studies have been done to investigate the quality of questions and answers on Q&A platforms [69, 79, 91, 102]. Sillito *et al.* [79] performed an empirical study on factors that make a good source code example on Stack Overflow. They found that explaining important elements and presenting a solution step-by-step make a good example. Treude *et al.* [91] performed a study on Stack Overflow to explore which questions are answered well and which ones remain unanswered. They found that source code is an important factor for "code review" questions to get a good answer. Ponzanelli *et al.* [69] studied the factors that potentially affect the

quality of questions on Stack Overflow. They showed that the attached source code is an important factor for question quality. In this chapter we study whether developers tend to reuse source code from a high-quality answer (i.e., a high-voted answer).

Figure 3.1: An example of a question and its accepted answer on Stack Overflow.

3.2.3 Source Code Reuse from Stack Overflow

Source code reuse can be commonly observed [6]. Listing 3.1 shows an example of code reuse. This source code snippet, which is taken from a GitHub project[1], is reused from the Stack Overflow post shown in Figure 3.1.

To help developers reuse source code, several approaches have been proposed. Rigby and Robillard [72] proposed an approach to extract code elements from various documents such as Stack Overflow posts. They evaluated their approach on 188 Stack Overflow posts containing 993 code

[1] https://goo.gl/X84SFi

Listing 3.1: Source code reuse example.

```
1 //http://stackoverflow.com/questions/2419749/get-
2 // selected-elements-outer-html
3 jQuery.fn.outerHTML = function (s) {
4   return s
5     ? this.before(s).remove()
6     : jQuery("<p>").append(this.eq(0).clone()).html();
7 };
```

elements. Their technique achieved an average 0.92 precision and 0.90 recall. Ponzanelli *et al.* [66] proposed an Eclipse plugin named Seahawk that helps developers search and import code snippets from Stack Overflow. Then they proposed an Eclipse plugin named Prompter which automatically searches and identifies Stack Overflow discussions, evaluates their relevance based on the given the code context in the IDE, and notifies the developer if a user-defined confidence threshold is surpassed [67, 68]. Armaly and McMillan [9] presented a novel reuse technique that allows programmers to reuse functions from a C or C++ program, by recording the state of the dependencies during one program's execution, and replaying them in the context of a different program.

Different from prior studies which focused on proposing approaches to retrieve code for developers to reuse, we are interested in studying how developers utilize the source code from Q&A platforms and which barriers they face when doing so.

3.2.4 Code Licensing on Stack Overflow

Understanding the license of a source code snippet is an important part of code reuse. Developers need to adhere to certain licenses (e.g., MIT and CC BY-SA 3.0) when reusing code from Q&A platforms. For example, developers may copy-and-paste source code from Stack Overflow posts into their own projects as long as they adhere to the Creative Commons Attribute-ShareAlike (CC BY-SA 3.0) license[2], according to an official Stack Overflow blog post [10]. One of the requirements of this license is that attribution is needed from the developers by putting a link to the original Stack Overflow post in their source code comments.

Significant number of studied have been performed on code licensing. An *et al.* [6] studied whether developers respect license restrictions when reusing source code from Stack Overflow in Android apps or vice versa. With a case study of 399 Android apps, they found 232 code snippets in 62 Android apps which were potentially reused from Stack Overflow,

[2] https://creativecommons.org/licenses/by-sa/3.0/

while 1,226 Stack Overflow posts contained code from 68 Android apps. In total, An *et al.* [6] observed 1,279 potential license violations. Almeida *et al.* [3] performed a survey among developers on open-source licenses and found that developers struggle when multiple licenses were involved. The results indicate a need for tool support to help guide developers in understanding this critical information about the license that is attached to software components. In this study, we would like to understand whether developers are aware about the code license of Stack Overflow; if developers are not, whether such unawareness forms a barrier for code reuse from Q&A platforms for developers.

3.3 Research Questions & Data Collection

In this section, we present our research questions and their motivation. In addition, we describe how we collected the datasets that we use to answer our research questions.

3.3.1 Research Questions

We focus on the following five research questions. The first two research questions are answered through an exploratory study on code reuse, in which we collect empirical evidence about code reuse from Stack Overflow in open-source projects. The last three research questions are answered through a survey, in which we contact developers of open-source projects who are active on Stack Overflow as well.

RQ1: To what extent do developers need to modify source code from Stack Overflow in order to make it work in their own projects?

Prior research has proposed several ways to utilize the source code from Stack Overflow [66–68, 72]. For example, Ponzanelli *et al.* [66] presented an approach to automatically construct queries from the current context in the Eclipse IDE and retrieve relevant code and its corresponding discussions from Stack Overflow. However, there is no empirical evidence about the process of how developers are reusing the source code, e.g., whether they copy-and-paste the original source code without any modification or they need to modify it considerably. Knowing how developers reuse source code from Q&A platforms will give us insights on how to make source code reuse easier in next-generation Q&A platforms.

RQ2: From which part of the Stack Overflow post does the reused source code come?

Intuitively, we may expect that accepted answers, or answers with a high score are the most useful. In this RQ, we study whether this intuition is correct. In particular, we investigate why developers chose to reuse code from non-accepted or low-scored answers. By understanding why developers chose non-accepted or low-scored answers, we can provide insights to help Q&A platforms organize their answers better so that developers can find solutions more easily.

RQ3: Do developers prefer reusing or reimplementing source code?

Intuitively, reusing source code will consume less effort than reimplementing, especially if the source code is well-tested. In this RQ, we survey developers about the correctness of this intuition. We also investigate which factors make developers prefer reimplementing source code over reusing it.

RQ4: Is code license a barrier for code reuse for developer?

A prior study shows that the amount of code reuse from Q&A platforms (i.e., Stack Overflow) is low (1.70%) [1]. One possible reason for this low percentage is that developers might not attribute the Q&A platforms from which a source code snippet comes (even though required by its license). Another possible reason is that some Q&A platforms (e.g., Stack Overflow) have a relatively restrictive code license, which might hinder developers from reusing source code in their own projects. Hence, in this RQ, we survey developers about their knowledge and understanding of the code licenses of Q&A platforms, to find out whether code license forms a barrier for code reuse from Q&A platforms for developers.

RQ5: How can code reuse be improved in next-generation Q&A platforms?

In this RQ, we elicit suggestions from the surveyed developers for improving next-generation Q&A platforms. We analyze and synthesize their suggestions to define a roadmap for researchers and developers of next-generation Q&A platforms.

3.3.2 Data Collection

In the remainder of this section, we describe our data collection process for the exploratory study (to answer RQ1 and RQ2) and our survey (RQ3–RQ5). All our studied data and the corresponding analysis are available from our online appendix [111].

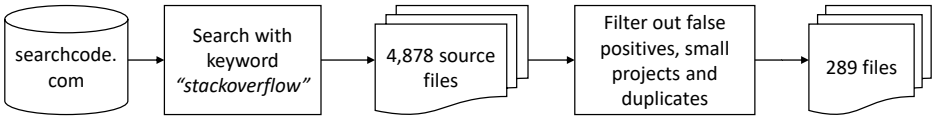

Figure 3.2: An overview of our data collection of the exploratory study.

Collecting Data for the Exploratory Study

The steps of our data collection process for our exploratory study on code reuse from Stack Overflow in open-source projects (Section 3.4) are shown in Figure 3.2. First, we collected source files that contain an explicit reference to a Stack Overflow post from `searchcode.com` [78], a source code search website. Then, we removed irrelevant files such as false positives and duplicate files.

Collecting Source Code Files from Open-Source Project Repositories: As explained in Section 3.2.4, developers must cite a Stack Overflow post when they reuse code or ideas from that post. Hence, to obtain source files that contain a source code snippet that is reused from a Stack Overflow post (either a question or an answer), we search for source files that contain at least one hyperlink to a Stack Overflow post.

To search for such source files, we use `searchcode.com` [78] as our search engine. `searchcode.com` has indexed over 20 billion lines of source code from 7 million open-source projects. With its API [77], we were able to collect 4,878 files in total using *"stackoverflow"* as the search keyword. We focused on files that are written in the five most popular programming languages on Stack Overflow [85] (JavaScript, Python, Java, PHP, and Objective-C).

Removing Irrelevant Files: The *"stackoverflow"* keyword can match source files that contain the *"stackoverflow"* keyword outside of a link to a post, e.g., in an API name. We manually removed such false positives in this step. In addition, not all projects in open-source repositories are interesting from a software engineering point of view. For example, GitHub contains many toy projects, from which we cannot extract knowledge that is representative of other projects [47]. Therefore, to mitigate the effects from small projects, we removed files that belong to projects with less than 1,000 commits and 10 contributors. We then removed the files that are duplicates of each other (e.g., because they come from forked projects). Finally, we ended up with 289 unique files, which belong to 182 open-source projects. Within these files, 321 Stack Overflow hyperlinks were found. Figure 3.3 shows the distribution of the studied Stack Overflow links over the five studied programming languages.

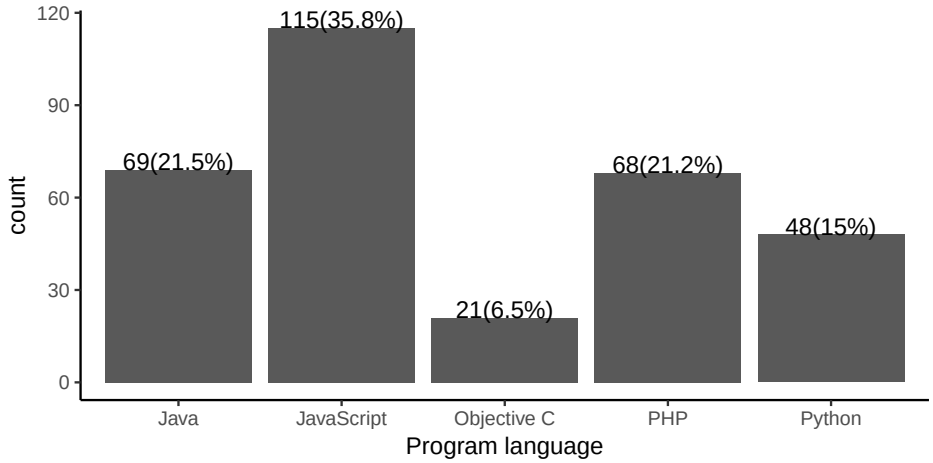

Figure 3.3: The distribution of the studied Stack Overflow links over the five programming languages.

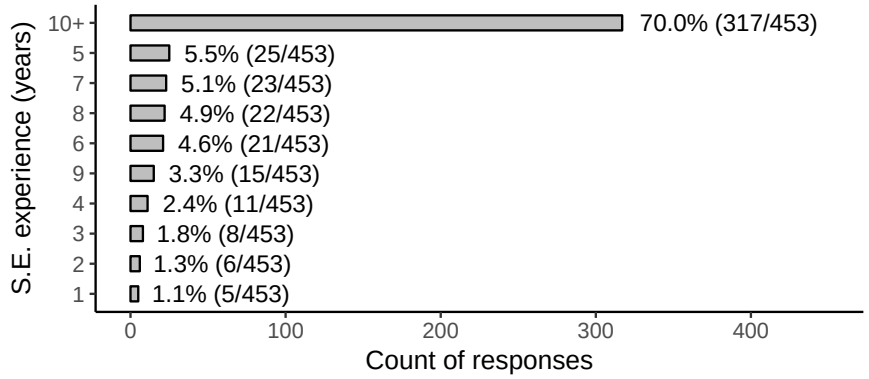

Figure 3.4: Distribution of the software engineering experience of the participants in years.

Collecting Participants for our Survey

We used the dataset provided by Vasilescu *et al.* [93] to get candidate participants. This dataset includes 93,771 email addresses from the intersection of users of GitHub and Stack Overflow. We took a random sample of 6,000 users from this dataset and sent them email invitations for our online survey. 1,935 of the emails did not reach the survey candidates because the email address did not exist any more. In the end, we received 453 responses which equals a response rate of 11.1%.

Figure 3.4 shows that 87.9% of the participants are experienced software engineers with more than 5 years experience. Industrial, open-source, and personal projects are the dominant project types that the participants are

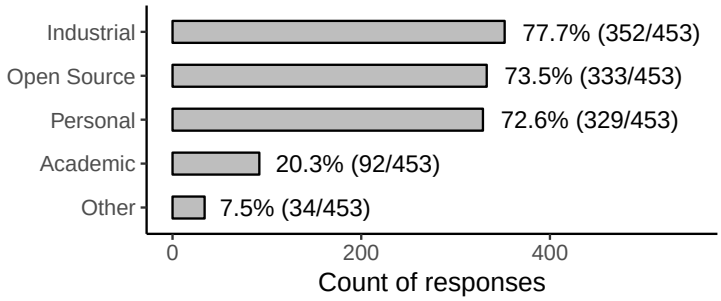

Figure 3.5: Distribution of the types of projects that the participants are working on.

involved in, followed by academic projects (see Figure 3.5). Note that a participant can work on more than one type of project.

3.4 An Exploratory Study of Source Code Reuse from Stack Overflow in Open-Source Projects

In this section, we present and discuss the results of our exploratory study of 321 Stack Overflow links in 289 source code files of 182 open-source projects. For each research question in our exploratory study, we discuss the used approach and results.

3.4.1 RQ1: To What Extent Do Developers Need to Modify Source Code From Stack Overflow in Order to Make It Work in Their Own Projects

Approach: To understand how developers utilize source code from Stack Overflow, we manually analyzed the collected source code and the referenced Stack Overflow posts. We manually extracted and categorized the type of code utilization from Stack Overflow posts for each collected source file. We performed a lightweight open coding-like process [75, 76] for identifying the type of the code utilization. This process involved 3 phases and was performed by the me and two research collaborators (i.e., P1–P3) of this study:

- Phase I: P1 extracted a draft list of types of source code utilization from Stack Overflow based on 50 source files and the linked Stack Overflow post. Then, P1 and P2 use the draft list to categorize the same source file collaboratively, during which the types were revised and refined. At the end of this phase, we obtained five types of source code utilization.

- Phase II: P1 and P2 applied the resulting types of Phase I to independently categorize all 289 collected source files. They took notes regarding the deficiency or ambiguity of the types for categorizing certain source files.
- Phase III: P1, P2, and P3 discussed the coding results obtained in Phase II to revolve the disagreements until a consensus was reached. No new types were added during this discussion. The inter-rater agreement of this coding process had a Cohen's kappa of 0.91.

Table 3.1 shows the final categorization of the types of source code utilization from Stack Overflow. In our study, one source code file-Stack Overflow post pair could only be categorized as one type. We did not run into conflicts because of this limitation.

Table 3.1: The identified types of source code utilization from Stack Overflow.

ID	Name	Definition	Count	Perc.
C1	Exact Copy	Developers copy-and-pasted source code from Stack Overflow **without any modification**.	66	20.5%
C2	Cosmetic Modification	Developers copy-and-pasted source code from Stack Overflow with modifications which **do not alter the functionality** of that source code (e.g., renaming identifier names to make it more readable).	32	10.0%
C3	Non-cosmetic Modification	Developers copy-and-pasted source code from Stack Overflow with modifications which **alter the functionality** of that source code (e.g., adding arguments to a function prototype).	69	21.5%
C4	Converting Ideas	Developers did not copy-and-paste any source code from Stack Overflow. Instead, they wrote the source code from scratch by **applying the ideas** in the answers.	40	12.5%
C5	Providing Information	Developers did not reuse any source code from Stack Overflow. Instead, they treated the Stack Overflow post as an **information source** related to the issue they are addressing.	114	35.5%

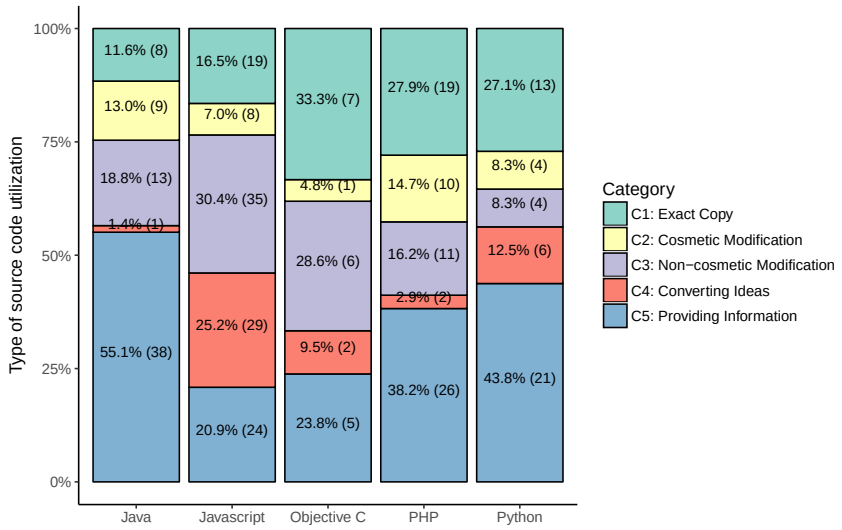

Figure 3.6: The distribution of each type of source code utilization for each of the studied programming languages.

Results: **31.5% of the reused source code was modified in one way or another.** Table 3.1 shows that 20.5% of the studied files reused source code without modification (C1). In 31.5% (C2 and C3) of the files, the source code required modification before it could be used. Type C4 (12.5%) indicates that it is not exceptional that developers converted the ideas written in natural language to source code from scratch. Type C5 (35.5%) indicates that there exist developers who use Stack Overflow as a "programming manual". The finding that 31.5% of the source code reuse required additional modification implies that finding the code is only the first step for code reuse. More effort is needed to facilitate code reuse from Q&A platforms after retrieving relevant code from them, such as making the source code work in the required context. Prior studies have addressed the problem of integrating source code in a target context automatically [4, 24, 105]. Our findings provide empirical support for the importance of such studies, and suggest that it may be promising to integrate the proposed code integration techniques into Q&A platforms.

In 10.0% of the studied files, developers make cosmetic modifications when reusing source code, which may improve the readability or simplicity of the source code. In the Cosmetic Modification category, developers copy-and-paste the source code from a Stack Overflow post and make modifications to the source code which may not be necessary

Listing 3.2: Source snippet from the project.[3]

```
hours = TimeUnit.MILLISECONDS
  .toHours(elapsedTimeMilliseconds);
minutes = TimeUnit.MILLISECONDS
  .toMinutes(elapsedTimeMilliseconds
  - TimeUnit.HOURS.toMillis(hours));
seconds = TimeUnit.MILLISECONDS
  .toSeconds(elapsedTimeMilliseconds
  - TimeUnit.HOURS.toMillis(hours)
  - TimeUnit.MINUTES.toMillis(minutes));
```

Listing 3.3: Source snippet from the Stack Overflow answer.[4]

```

final long hr = TimeUnit.MILLISECONDS.toHours(l);
final long min = TimeUnit.MILLISECONDS
  .toMinutes(l - TimeUnit.HOURS.toMillis(hr));
final long sec = TimeUnit.MILLISECONDS
  .toSeconds(l - TimeUnit.HOURS.toMillis(hr)
  - TimeUnit.MINUTES.toMillis(min));
```

to make the source code work in the target project. In the example shown in Listing 3.2 and 3.3, the developer copied three lines of source code in the accepted answer from the Stack Overflow post and renamed the variable name from `hr`, `min`, and `sec` to `hours`, `minutes`, and `seconds`, respectively.

In 12.5% of the files, developers wrote the source code from scratch based on the descriptions of the algorithm. In the example shown in Listing 3.4 and Listing 3.5, developers implemented a function to detect whether a line intersects with a rectangle (Listing 3.4), based on an answer from the Stack Overflow post shown in Listing 3.5.

Another example is shown in Listing 3.6 and 3.7, where the developers wrote a regular expression that extracts all Youtube video ids in a string (see Listing 3.6). This source code snippet was modified based on the source code from the Stack Overflow post shown in Listing 3.7, which was written in PHP. In this example, developers actually rewrote the regular expression in JavaScript based on the PHP source code from the Stack Overflow post. We categorized this file under the Converting Ideas type since developers

[3] https://goo.gl/9ouSz1
[4] https://goo.gl/74oVBu
[5] https://goo.gl/4ezMUr
[6] https://goo.gl/1Wn9vF
[7] https://goo.gl/HK5kyV
[8] https://goo.gl/eq1Dnk

Listing 3.4: Type C5: An example of converting descriptions into source code: source snippet from the project[5] is implemented based on the description of the algorithm from Stack Overflow (see Listing 3.5).

```
Rect.prototype.collideLine = function(p1, p2) {
  var x1 = p1[0];
  var y1 = p1[1];
  var x2 = p2[0];
  var y2 = p2[1];

  function linePosition(point) {
    var x = point[0];
    var y = point[1];
    return (y2-y1)*x + (x1-x2)*y + (x2*y1-x1*y2);
  }

  var relPoses = [[this.left, this.top],
              [this.left, this.bottom],
              [this.right, this.top],
              [this.right, this.bottom]
             ].map(linePosition);

  var noNegative = true;
  var noPositive = true;
  var noZero = true;
  relPoses.forEach(function(relPos) {
    if (relPos > 0) {
      noPositive = false;
    } else if (relPos < 0) {
      noNegative = false;
    } else if (relPos === 0) {
      noZero = false;
    }
  }, this);

  if ( (noNegative || noPositive) && noZero) {
    return false;
  }
  return !((x1 > this.right && x2 > this.right) ||
        (x1 < this.left && x2 < this.left) ||
        (y1 < this.top && y2 < this.top) ||
        (y1 > this.bottom && y2 > this.bottom)
        );
};
```

Listing 3.5: Description of the algorithm in the Stack Overflow answer.[6]

```
Let the segment endpoints be p1=(x1 y1) and p2=(x2 y2).
Let the rectangle's corners be (xBL yBL) and (xTR yTR).

Then all you have to do is

A. Check if all four corners of the rectangle are on the
same side of the line. The implicit equation for a line
through p1 and p2 is:

F(x y) = (y2-y1)x + (x1-x2)y + (x2*y1-x1*y2)

If F(x y) = 0, (x y) is ON the line.
If F(x y) > 0, (x y) is "above" the line.
If F(x y) < 0, (x y) is "below" the line.

Substitute all four corners into F(x y). If they're all
negative or all positive, there is no intersection. If
some are positive and some negative, go to step B.

B. Project the endpoint onto the x axis, and check if the
segment's shadow intersects the polygon's shadow. Repeat
on the y axis:

If (x1 > xTR and x2 > xTR), no intersection (line is to
right of rectangle).
If (x1 < xBL and x2 < xBL), no intersection (line is to
left of rectangle).
If (y1 > yTR and y2 > yTR), no intersection (line is
above rectangle).
If (y1 < yBL and y2 < yBL), no intersection (line is
below rectangle).
else, there is an intersection. Do Cohen-Sutherland or
whatever code was mentioned in the other answers to
your question.

You can, of course, do B first, then A.
```

Listing 3.6: Source snippet from the project in JavaScript.[7]

```
YOUTUBE_REGEXP: new RegExp(
  '(?:https?://)?' + // Optional scheme. Either...
  '(?:www\\.)?'    + // Optional www subdomain
  '(?:'            + // Group host alternatives
  'youtu\\.be/'    + // Either youtu.be,
  [...]
  ')'        // End negative lookahead assertion.
),
```

Listing 3.7: Source snippet from the Stack Overflow answer in PHP.[8]

```
1  //Linkify youtube URLs which are not already links
2  function linkifyYouTubeURLs($text) {
3    $text = preg_replace('~(?#!js YouTubeId Rev:...
4      # Match non-linked youtube URL in the wild...
5      https?://          # Required scheme...
6      (?:[0-9A-Z-]+\.)?  # Optional subdomain.
7      (?:                # Group host alternatives.
8      youtu\.be/         # Either youtu.be,
9      [...]
10   $text);
11   return $text;
12 }
```

cannot reuse the source code directly from another language, instead, they have to convert the idea and rewrite it from scratch.

Developers used Stack Overflow posts in 35.5% of the files as an information source for later reference. In 35.5% of the files, developers did not reuse any source code from Stack Overflow. Instead, they put a Stack Overflow hyperlink in their source code to provide background information about the issue or solution. For example, there is a file[9] in which the developer gave a warning that the usage of dict can be dangerous if multiple headers are set in the Set-Cookie header and the developer also provided the link to the Stack Overflow post which discussed this issue in the source code.

Developers are the most likely to reuse code or ideas in JavaScript. Figure 3.6 shows the distribution of each type of source code utilization for each studied programming language. We observe that code and idea reuse was the highest in JavaScript (79.1% of the studied JavaScript files). One possible explanation is that Stack Overflow provides an online running environment for JavaScript, which may make developers more confident about reusing code or ideas in JavaScript from Stack Overflow than in other languages.

31.5% of the reused source code required additional modification, which shows the importance of studies on automatic code integration. In 12.5% of the studied files, developers reimplemented code based on an idea, which suggests that Q&A platforms should consider to summarize the key points that are discussed in a post to give developers a quick view of the question and its answers.

[9] https://goo.gl/KKbPWk

Table 3.2: Where does the reused source code come from?

Source	HV*	NHV**	Total	Perc.
Accepted Answer	144	11	155	48%
Non-Accepted Answer	35	48	83	26%
Question	-	-	5	2%
NOT REUSE	-	-	78	24%
Total	-	-	321	100%

* Highest-voted answers
** Non-highest-voted answers

3.4.2 RQ2: From Which Part of the Stack Overflow Post Does the Reused Source Code Come?

Approach: We manually inspected from which part (e.g., accepted answer, non-accepted answer, or question) of the Stack Overflow post the reused source code originates. We also check whether the answer is the highest-scored one. Two of the collaborators manually examined each source code file and the linked post (including the question, all answers, and all comments to the answers) individually and categorized it. Discrepancies were discussed until a consensus was reached. The discrepancies were due to the difficulty of identifying the exact answer that was reused (in particular, when only the idea of a code snippet was reused). After identifying the reused answer, the categorization was straightforward. The inter-rater agreement of this categorization had a Cohen's kappa of 0.85.

Results: **In 26% of the studied files developers chose a non-accepted answer and in 58%, those non-accepted answers were not the highest-scored ones.** The results of the categorization are shown in Table 3.2. As we can see from the results, not all reused source code came from an accepted answer. In 48% of the studied files, developers chose source code from an accepted answer. However, there are still a considerable number (26%) of files where developers choose the source code from non-accepted answers. Moreover, among those non-accepted answers, 58% were not the highest-scored ones, which indicates that developers certainly did not always choose source code from the accepted or highest-scored answer. In the remainder of this section, we discuss the situations in which developers reused source code from a non-accepted answer in more detail.

Different Requirements than the Question Asker

[10] `https://goo.gl/Z1pRMS`

Listing 3.8: Source snippet in the project that implements a method to generate GUIDs.[10]

```
1 // http://stackoverflow.com/questions/105034/how-to-
2 // create-a-guid-uuid-in-javascript
3 function generateID() {
4   return "avalon"
5     + Math.random().toString(36).substring(2, 15)
6     + Math.random().toString(36).substring(2, 15)
7 }
```

Listing 3.9: Source snippet in the accepted answer on Stack Overflow.[12]

```
1 function guid() {
2   function s4() {
3     return Math.floor((1 + Math.random())
4       * 0x10000).toString(16)
5       .substring(1);
6   }
7   return s4() + s4() + '-' + s4() + '-' + s4()
8     + '-' +s4() + '-' + s4() + s4() + s4();
9 }
```

Description: Developers chose source code from a non-accepted answer because they had different requirements than the original question asker.

Example: A developer wanted to implement a method to generate GUIDs. The source code in this example is shown in Listing 3.8. This source code snippet is actually from a non-accepted answer[11] on Stack Overflow which has 37 votes, while the accepted answer has 1290 votes. The source code provided by the accepted answer is shown in Listing 3.9.

According to the description in the answer that contains the source code, the algorithm in Listing 3.8 is simpler and has very good performance, but not compliant with the RFC 4122 standard. The author of this answer also attached a performance test result in which several algorithms that are mentioned in other answers of the Stack Overflow post are compared, which shows that the algorithm in Listing 3.8 outperforms the others. Hence, one possible explanation is that the developer who adopted this low-scored answer prioritizes performance and simplicity over other factors, such as whether the generated result is compliant with a standard.

[11] https://goo.gl/aC4auZ
[12] https://goo.gl/xpAcga

Fixing Bugs

Description: Developers adopted source code that improves on the accepted answer (e.g., by fixing a bug, handling additional cases).

Example: A developer was looking for a method to draw a dashed line around a selection area in JavaScript[13]. The non-accepted answer[14] improves the accepted answer by utilizing the built-in transformation functionality of Canvas, and also handles special cases where the line is vertical, which was not addressed in the accepted answer.

Improving Speed

Description: Developers adopted source code with a better performance.

Example: A developer was looking for an algorithm that sorts an array by the Levenshtein Distance in JavaScript. According to the comments below the accepted answer, the implementation in the accepted answer performed better than the one provided by the original asker. However, a non-accepted answer provided an improved version of the accepted answer which was described as *"Most speed was gained by eliminating some array usages"*, which was reused by the developers in their project. Thus we believe this developers gave performance a higher priority.

Unsurprisingly, we found that developers have different requirements for their solutions. Even if answers that are provided in the post do not meet the requirements of the asker, other developers may find them useful (e.g., a solution with higher performance). For developers who are looking for solutions on Stack Overflow, it is better to go through all the answers of a relevant question instead of focusing on the accepted answers. Q&A platforms should improve the way of organizing answers, so that developers can find the most suitable answers based on their requirements faster. For example, Q&A platforms may allow users to vote on different aspects, such as the readability or performance of the source code in an answer. The results from our user survey confirm the need for this improvement (see Section 3.6.2).

Developers reused code from a non-accepted or low-scored answer for various reasons, such as the simplicity and performance of the source code. Some even reused code from an answer that delivered a total opposite from what the asker wanted. Hence, Q&A platforms should improve the way in which answers are organized, so that developers can find the most suitable answers based on their requirements easily.

[13] `https://goo.gl/gzMCgy`
[14] `https://goo.gl/8foVXq`

3.5 A Survey on Code Reuse from Stack Overflow

Survey Design: Two of the research collaborators posited the survey questions that cover the three research questions (see Section 3.3.1). The third research collaborator checked the questions to eliminate any ambiguity from the wording of the survey. Before sending the survey to the 6000 participants that we collected in Section 3.3.2, we sent a draft version of the survey to 20 participants (excluded from the 6000 participants). We received feedback from seven of them, and refined the survey based on this feedback. The questions in the survey are available in the Appendix. The survey is divided into three parts:

1. **Demography (Q1 - Q7)**: these questions collect information about the software engineering background of the participants.

2. **Barriers (Q8 - Q17)**: these questions collect information about the barriers that the participants face when reusing source code from Q&A platforms. We included only the responses from participants who have ever reused source code from Q&A platforms (i.e., those who answered *yes* to Q7, which were 380 (83.9%) participants).

3. **Suggestions (Q18 - Q19)**: these questions collect suggestions for next generation Q&A platforms. Every participant could answer these two questions, regardless of whether they ever reused source code from Q&A platforms.

Data Analysis: The responses of the survey are available in our online appendix [111]. The survey contained 12 open-ended questions in which participants could choose to input their own responses in free-form text. For each of these questions, we used an open coding-like approach to let the coding schema emerge during the analysis [37]. We adopted a three phase coding process:

- Phase I: two of the research collaborators (P1 and P2) coded the responses of each open-ended question individually. As a result, both P1 and P2 had their own set of codes for the answers. Then, P1 and P2 discussed their draft code schema and made a revised version of the code schema.

- Phase II: P1 and P2 used the revised schema to code the answers. Then, they discussed and resolved conflicts. We use Cohen's Kappa [21] to measure the level of agreement. Values < 0 are characterized as indicating no agreement and 0âĂŞ0.20 as slight, 0.21âĂŞ0.40 as fair, 0.41âĂŞ0.60 as moderate, 0.61âĂŞ0.80 as substantial, and 0.81âĂŞ1 as almost perfect agreement. The Cohen's Kappa value for our coding result was 0.92, which indicates an almost perfect agreement. As

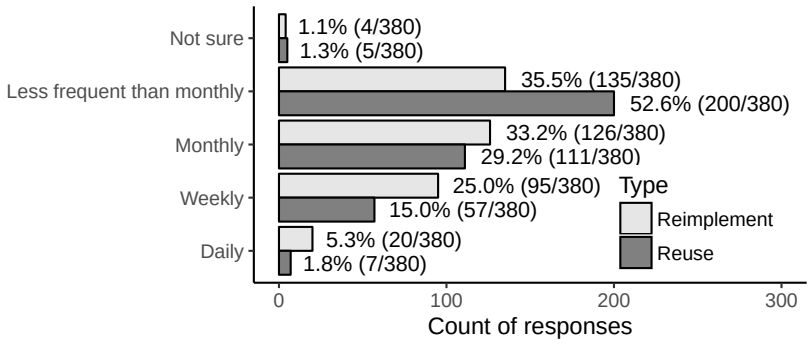

Figure 3.7: Comparison of frequency of reusing and reimplementing source code.

a result, a unified coding schema was developed and applied to all the answers.

- Phase III: three of the research collaborators (P1, P2 and P3) discussed the coding results obtained in Phase II to revolve disagreements until a consensus was reached. The interrater agreement of this coding process had a Cohen's Kappa of 0.79, which indicates a moderate agreement.

We answer RQ3 and RQ4 in the remainder of this section and RQ5 in Section 3.6.

3.5.1 RQ3: What Are the Preferences of Developers When It Comes to Reusing Code?

Developers reimplement source code slightly more frequently (i.e., for daily, weekly, and monthly cases) than that they reuse source code. Figure 3.7 shows the comparison of frequency of reimplementing source code and reusing source code from Q&A platforms. The number of participants who reimplement source code monthly (33.2%) and those who reuse source code monthly (29.2%) are close, while the difference increases to 25.0% vs. 15.0% at a weekly frequency.

A majority of developers (65%) prefer reimplementing source code, due to the code modification that is required to make the code from the post work in their own project. Table 3.3 shows the reasons for choosing reimplementation over the reuse of source code. The top reason that makes developers prefer reimplementing source code is the code modification that is required to make the code from the post work in their own projects. This finding is consistent with our finding in RQ1 (i.e., most code needs modification before reusing) and also provides empirical

Table 3.3: Reasons for choosing reimplementing over reusing source code. (Multi-selection allowed, hence the sum of the percentages is larger than 100%.)

Category	**Description**	**Perc.**
Context	The code should be written according to its context.	65%
Comprehension	Do not understand the source code to be reused.	44%
Quality	The quality of the source code is too low.	32%
Time consuming	Reusing source code takes more time.	17%
Other	Other reasons.	7%

evidence for the importance of research on code integration. Several studies have been done on automatically retrieving and integrating code in a user's project context [4, 28, 105]. However, these approaches are not widely adopted by developers. Future studies should investigate which factors prevent such tools from being applied in practice.

Code comprehension ranks as the second most important reason that preferring reimplementation over code reuse. This finding is in line with the work by Xin *et al.* [113], which showed that developers spend 58% of their time on program comprehension activities. Hence, next-generation Q&A platforms should investigate how to improve comprehension of the source code in a post to facilitate its reuse. Approximately one-third (32%) of the participants complained about the low code quality on Stack Overflow, which highlights the need for next-generation Q&A platforms to improve or verify the code quality of source code snippets.

An interesting observation was that 17% of the participants stated that reusing source code takes more time than reimplementing it, which is against the common wisdom. One possible reason is that if the source code snippet is large or complex, it could take more time to comprehend it than to make it work in another context.

Developers reimplement source code slightly more frequently than that they reuse source code. The primary reason is that it would take longer to adapt the source code to work in their own projects, than to simply reimplement it. Our observations provide empirical evidence for the importance of research on automated code integration and code comprehension, and highlight the need of improving the quality of code snippets on next-generation Q&A platforms.

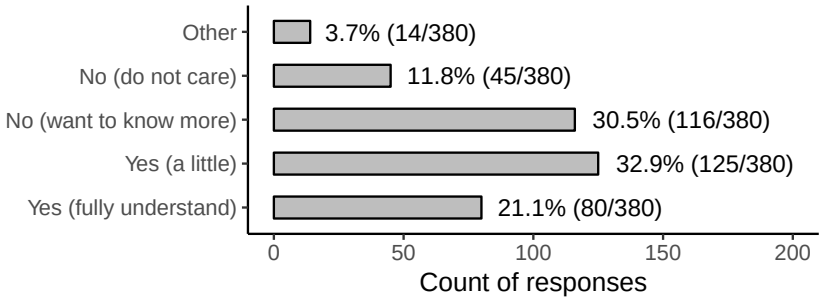

Figure 3.8: Participants' awareness of the licenses of Q&A platforms.

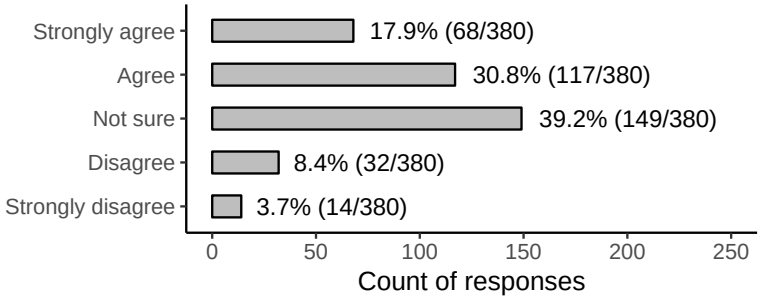

Figure 3.9: Participants' opinion about license compatibility between Q&A platforms and their projects.

3.5.2 RQ4: Is Code License a Barrier for Code Reuse for Developers?

In 75.2% of the cases, participants do not have a good understanding of the license terms of Q&A platforms, which indicates that there may be license violation issues when developers reuse source code from Q&A platforms. Figure 3.8 shows the results of participants' awareness of the licenses of Q&A platforms. An *et al.* [6] studied code reuse on Android apps and observed 1,279 potential license violation cases where developers reused source code from Q&A platforms in Android apps, or vice versa. Our survey results give a possible explanation for such violations. The *"Other"* category in Figure 3.8 includes cases in which the participants did not give a concrete answer, e.g., *"Depends on the platform. Stack Overflow is attribution-required, but the requirements of most other sites are vague or not generally known."*

In 39.2% of the cases, participants are not sure whether the license of a Q&A platform is compatible with that of their own project. Figure 3.9 shows that an additional 12.1% of the participants (strongly) disagrees that the license of a Q&A platform is compatible with

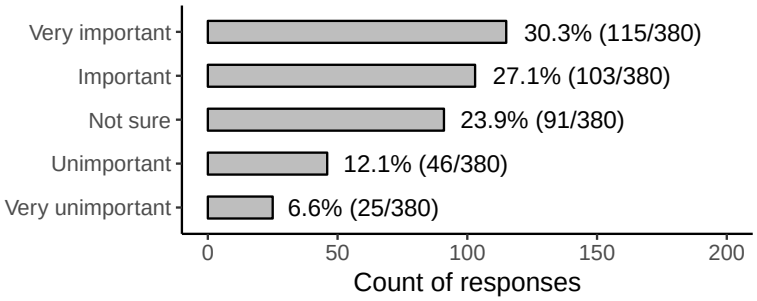

Figure 3.10: Importance of having more information on license.

that of their own project. Hence, 51.3% of the participants may experience difficulties (or cause a violation) when reusing code from Q&A platforms due to their license. These difficulties were noticeable from the survey responses when the participants were asked why they preferred reimplementing over reusing source code. For example, one participant mentioned that *"licensing is sometimes an issue."*.

More than half of the participants (57.4%) think that having more information about the code license is (very) important (see Figure 3.10). Together with the finding that most participants do not have a good understanding of the code license of Q&A platforms, these findings reveal a need for clearer information about the code license of Q&A platforms.

Generally speaking, participants did not have a good understanding of the code license on Q&A platforms. More than half of the participants believed that, or were unsure whether, there exist incompatibilities between the code license of their own project and Q&A platforms. Almost 60% of the participants thought that Q&A platforms should give more information about their code license. Based on these observations, next-generation Q&A platforms should have clearer license information and make that information more visible to developers.

3.6 A Roadmap for Next-Generation Q&A platforms

In this section, we summarize and analyze the results for RQ5 (How can code reuse be improved in next-generation Q&A platforms?). In total, we collected 150 responses for Q19 in the survey. 22 of these responses were not actually suggestions for next-generation Q&A platforms (e.g., *"Not much, quite happy with Stack Overflow."*) and were excluded from the following analysis. Each response can contain multiple suggestions. In total,

we extracted 183 suggestions from the responses. Using our open coding-like approach (see Section 3.5), each suggestion was categorized into one of these five categories: code quality, information enhancement & management, data organization, license, and human factors. We categorized suggestions that did not fall into one of these categories into a sixth "other" category. We highlight the findings and discuss implications on future research of next-generation Q&A platforms of each category in the remainder of this section.

Table 3.4: The categorization of code quality suggestions — 64 out of 183 (35.0%).

Category	Description (D) – Example (E)	Perc.	Count
Integrated validator	**D:** Integrated validator that can test the code snippets on Q&A platforms. **E:** *"An inbuilt REPL environment for as many languages/environments as possible."*	42.2%	27
Outdated code	**D:** Answers (including source code) on Q&A platforms suffer from out-of-date problems. Participants are seeking for a solution to this problem. **E:** *"make date important in marking outdated code, and deprecate those snippets via the community"*	29.7%	19
Answer quality	**D:** Classifier that helps distinguish high and low quality answers. **E:** *"Better support for answers that are good, but out of date."*	17.2%	11
Code review	**D:** Integrated code review tool that helps improve the code quality. **E:** *"In-browser code review and commenting similar to that provided by commercial code review tools."*	10.9%	7

3.6.1 Suggestions on Code Quality

Next-generation Q&A platforms should integrate mechanisms for online code validation and detecting outdated code. Code quality is the most popular type of suggestion (35.0%) from the participants. Table 3.4 shows the categorization of the suggestions that participants made on improving the code quality on Q&A platforms. The two most important suggestions from developers on improving code quality were adding (1) an integrated validator (27 participants) that can test source code online and (2) an outdated code detection mechanism (19 participants) that can identify code for old software versions.

An integrated validator is a convenient way of testing source code snippets online to ensure the quality of the source code. Participants described such a tool for example as follows: *"The ability to interact with and run the code examples written in answers and questions"*. Several Q&A platforms have started to integrate online validation into their websites. For example, Stack Overflow can validate three web-languages: HTML, CSS, and JavaScript [84]. However, Stack Overflow does not support online validation of other languages, such as Java and C++, which are also very popular on Stack Overflow. There are several challenges when it comes to online validation of all languages.

One of the biggest challenges is to make an incomplete code snippet run correctly, since code snippets on Q&A platforms are usually not minimal working examples (MWEs). Often the answerer only needs to implement the core part of a solution and may leave out necessary context information (e.g., the required software version). To address this problem, prior studies have proposed several approaches to extend incomplete code snippets (not limited to those on Q&A platforms) into compilable ones based on program analysis and machine learning techniques [65, 70, 100]. However, none of these approaches can guarantee the correctness of the extended code. For example, there is a function called "foo" in a code snippet. To make this code snippet compilable, Q&A platforms need to infer where this function comes from and then import the corresponding library. However, it is difficult to automatically infer the exact library based on the source code snippet only. This problem may be solvable in Q&A platforms by leveraging the description that comes with the source code. Hence, future research should investigate whether the description of the source code can be used to improve the correctness of the automatic code extension.

Providing an outdated code detection mechanism is the second most popular suggestion in the code quality category. Many of the participants mentioned that source code on Q&A platforms is often outdated and not suitable for current technologies or situations. For example, one participant suggested: *"Have explicit mechanisms for dealing with content that goes out of date due to platform or language changes."* Some participants

suggested a mechanism that clarifies the API version of the source code: *"Clear associates between the code snippets the versions of the API under which it will work. This is particularly when working with APIs that change frequently, like iOS and Unity."* Some also suggested deprecating outdated answers: *"Make date important in marking outdated code, and deprecate those snippets via the community."* This problem was recognized by various developers on Stack Overflow [52] and received wide attention from the Stack Overflow communities.

However, as far as we know, no existing study has investigated outdated source code or solutions on Q&A platforms so far. Hence, there is a need for future research on developing mechanisms to deal with outdated source code or solutions. There are two primary directions to deal with outdated code or solutions. First, future studies should propose approaches to automatically identify *outdated source code* or solutions in Q&A platforms. Second, future research should investigate how *incentive systems* can motivate communities to identify and update outdated source code or solutions.

Table 3.5: The categorization of information enhancement & management suggestions — 43 out of 183 (23.5%).

Category	**Description (D) – Example (E)**	**Perc.**	**Count**
Answer tagging	**D:** Better tagging-like information system for answers. **E:** *"Provide/require tagging of the version number(s) of the language [...]"*	37.2%	16
Code evolution	**D:** Better management of the evolution/revisions of code snippets. **E:** *"where does the code come from and copied to, and also the revisions inside the platform."*	14.0%	6
Resources linking	**D:** Q&A platforms should suggest for other resources (e.g., books, API documents, libraries etc.) **E:** *"Books suggestions based on questions."*	11.6%	5
Answer writing support	**D:** Support for writing better questions/answers. **E:** *"[...] it would be nice if it would be easier to ask a good question [...]"*	9.3%	4
Other	**D:** Other aspects of information enhancement & management. **E:** *"Built in support within an IDE to make it faster to get the answer you are interested in."*	27.9%	12

3.6.2 Suggestions on Information Enhancement & Management

Next-generation Q&A platforms should allow tagging for answers. Information enhancement & management (23.5%) is the second most popular suggestion for developers on Q&A platforms. Table 3.5 presents the results of suggestions related to information enhancement and management. Based on the observations, future studies should focus on recommending the tagging-like information for answers. These recommendations could be made automatically using, e.g., machine learning techniques [99, 101, 122], or aspect-mining techniques [56, 97, 109, 116, 121].

Table 3.6: The categorization of data organization suggestions — 21 out of 183 (11.5%).

Category	**Description (D) – Example (E)**	**Perc.**	**Count**
Code searching/indexing	**D:** Support for easier code search. **E:** *"Source Code indexing for easier retrieval. It could also give the possibility to find example of usage functions."*	47.6%	10
Duplicate posts	**D:** An automatic way of clustering duplicate questions/answers. **E:** *"Auto-suggest similar questions, particularly for questions that don't have answers."*	38.1%	8
Comments	**D:** Support on utilizing the comments of posts. **E:** *"Code in *comments* must be expressed better, than on Stack Overflow."*	14.3%	3

3.6.3 Suggestions on Data Organization

Next-generation Q&A platforms should better organize their data, for example by providing a better searching and indexing mechanism and better duplicate detection. As shown in Table 3.6, ten participants suggested that Q&A platforms should have a better way to index and search code. For example, one of the participants mentioned: *"Ability to search questions based on the version of the framework or language I'm working with"*. Eight participants suggested that Q&A platforms should have an automatic way to detect duplicate or similar posts and be organized in a better way. Three participants suggested to improve the utilization of the comments on posts.

In prior studies [12, 57, 62, 78, 98], researchers have studied code search engines to help developers to improve their search efficiency on source code. Our findings support these studies, and it would be interesting to integrate such code search engines into Q&A platforms.

Researchers proposed various approaches to help Q&A platforms detect duplicate questions automatically [2, 106, 119, 120]. The common way to identify duplicate questions is to measure such questions' similarity in terms of semantic meaning. Recently, deep learning has proven its power of capturing semantic meaning from natural language in several studies [16, 19, 30, 53]. Hence, future research could consider to employ deep learning to detect duplicate or find similar questions.

Table 3.7: The categorization of code license suggestions — 23 out of 183 (12.6%).

Category	**Description (D) – Example (E)**	**Perc.**	**Count**
Clearer license	**D:** Q&A platforms should make their license terms clearer. **E:** *"By far the most important requirement is clear licensing. Much of the code provided on such platforms is not currently usable because the license is unclear."*	69.6%	16
Permissive license	**D:** Q&A platforms should use a more permissive license. **E:** *"Let the user choose a more re-user-friendly license (e.g. copy without reference)."*	30.4%	7

3.6.4 Suggestions on Code License

Next-generation Q&A platforms should make their code licensing information clearer and more visible. In 12.6% of the cases, participants suggested to improve license-related issues, in particular to make the license more clear (16 participants). This percentage is in line with our earlier finding that 75.2% of the participants did not have a good understanding of the license terms of Q&A platforms (see Section 3.5.2). Table 3.7 shows the suggestions about the code license of Q&A platforms.

Participants requested that Q&A platforms provide a clearer explanation of their license terms: *"By far the most important requirement is clear licensing. Much of the code provided on such platforms is not currently usable because the license is unclear."* If developers would neglect the license of Q&A platforms and reuse source code from these platforms, they are under the risk of license violation which may cause legal problems later. The scale of license violation has been studied by [6]. An *et al.* [6] investigated code reuse in Android apps and observed a significant number of potential license violation cases when developers reused source code from Q&A platforms in Android apps, and vice versa.

Seven participants suggested that Q&A platforms should use a more permissive license, which has fewer restrictions on source code reuse. In the example of Stack Overflow, CC BY-SA 3.0 was the original license for the source code on this platform. CC BY-SA 3.0 is a copyleft (non-permissive) license which requires the derivative work to be licensed under the same license (CC BY-SA 3.0). This means that when developers reuse the source code from Stack Overflow into their projects, they have to license these projects under the CC BY-SA 3.0 license as well. Otherwise, they are under the risk of license violation.

It is also worth noting that, although Stack Overflow has announced this license change in a post on Stack Exchange [82], the change is not reflected on their homepage [86], which still says *"user contributions licensed under cc by-sa 3.0 with attribution required."* As such mismatches will further deepen developers' misunderstanding of license terms. Therefore, we suggest that next-generation Q&A platforms explicitly describe their license terms for source code reuse in a consistent manner.

Table 3.8: The categorization of human factor suggestions — 19 out of 183 (10.4%).

Category	**Description (D) – Example (E)**	**Perc.**	**Count**
Better curator	**D:** Better curators are needed to help improve the quality of the posts. **E:** *"Definitely curators for specific languages to rate answers in specific areas."*	63.2%	12
Gamification-related	**D:** Suggestions on improving the gamification system of Q&A platforms. **E:** *"Base reputation on number of answers up-voted by others, not on personal activity."*	36.8%	7

3.6.5 Suggestions on the Human Factor

Next-generation Q&A platforms should assign human experts to curate knowledge on the platform. In 10.4% of the cases, participants suggested to improve Q&A platforms in terms of the human factor. Table 3.8 shows that twelve participants suggested to have better curators to improve the quality of posts and help with marking good answers. One participant suggested: *"Pay some vetted, experienced developers to check the answers, instead of relying on gamification."* Another suggestion emphasized the importance of collaboration within the community: *"Arriving at a 'most correct' solution should be a more collaborative effort with a clearly shown path of how it was arrived at by multiple people, not necessarily just one user who takes all the credit."* Zagalsky *et al.* [117] revealed a shift within the R community from knowledge creation to knowledge curation. Hence, we suggest that next-generation Q&A platforms study other communities to improve the knowledge curation and collaboration processes.

Seven participants suggested to improve the gamification system of Q&A platforms. The usage of gamification on Q&A platforms has been proven effective before [7, 18]. However, participants revealed several flaws in this system. For example, one participant wrote: *"Base reputation on number of answers up-voted by others, not on personal activity. (Stack Overflow has too many nit-pickers gaining reputation by down-voting legitimate questions.)"*

Our findings suggest that future studies on how to improve the gamification mechanism of Q&A platforms are necessary.

Table 3.9: The categorization of other suggestions — 13 out of 183 (7.0%).

Category	Description (D) – Example (E)	Perc.	Count
Open source	**D:** Open sourced/community-hosted Q&A platforms. **E:** *"Open source to be competitive, self-hosted and easy to deploy (even without requiring docker or similar, to be usable in low end containers or even in hosting platforms)."*	30.8%	4
AI	**D:** Include AI-related techniques into the Q&A platforms. **E:** *"Let AI write code, we do code review."*	23.0%	3
Other	**D:** Cross-language support. Q&A platforms. **E:** *"When the clone is written in a different language, a language translation is automatically performed if not with some manual assistance"*	46.2%	6

3.6.6 Other Suggestions

Table 3.9 shows the 13 suggestions that did not fall into the other categories. There were four suggestions that suggested that Q&A platforms are open-sourced. In addition, there were three suggestions that were related to AI techniques. For example, one of them suggested the use of an AI technique that can automatically produce source code while developers only need to review the source code. These suggestions support the current research on code generation, in which tools are developed to automatically generate code based on a natural language input [41, 115]. The rest of the suggestions in the "other" category talk about different topics. For example, one of them suggests the integration of a tool that can automatically translate a source code snippet across programming languages.

In summary, based on the findings from our survey, we observe that the difficulty in fitting code in their own projects, a low comprehension and low quality of the code are the top barriers that prevent developers from reusing code. Lacking a good understanding of the code license is also a significant barrier for code reuse. Thus, future studies are encouraged to address these barriers to better facilitate code reuse for developers.

3.7 Threats to Validity

External validity. Threats to external validity relate to the generalizability of our findings. In this study, we used `searchcode.com` as our search engine and found 4,878 source files in total that contained links to Stack Overflow posts. After removing the small projects and duplicate files, there were 321 Stack Overfow links and 289 files left. The number of files may not be large enough to represent all the cases in the real world. The reason for restricting our study to these 321 Stack Overflow links is that we want to ensure that we study code reuse in serious software projects only. [47] showed that many open source projects are toy projects which do not adequately reflect software engineering practices. Hence, when studying open source projects, it is important to ensure those toy projects are removed from the data set. We ensure that such projects are removed by imposing strict selection criteria on our data (i.e., we only study files from projects that have at least 1000 commits and 10 contributors). We are confident that the studied number is large enough for our exploratory study to identify the core issues that are involved in the process of code reuse from Q&A platforms.

For the survey, we only invited developers who are in the intersection of users of GitHub and Stack Overflow. Hence, our results may not generalize to software developers who are not in this intersection, such as developers

of closed-source software. Future studies should extend our exploratory study and survey to developers from other domains.

We focused the first part of our study on the five most popular programming languages, and as a result, our findings may not generalize to other languages. Future work is necessary to investigate whether our findings hold for other languages.

Most of our findings are general in nature and seem to apply to code reuse in general. However, as we have no solid evidence, we cannot make definite claims about the generalizability of our findings outside Q&A platforms. Future studies should investigate whether our findings are valid outside Q&A platforms.

Internal validity. Threats to internal validity relate to the experimenter bias and errors. In this chapter, we heavily rely on manual analysis. For example, we manually inspected the source code in the projects and in Stack Overflow posts, we manually categorized each case of how developers are reusing the source code, and we manually categorized the survey responses. Unfortunately, these tasks are extremely difficult to automate. For example, it is very hard to automatically identify the Non-cosmetic Modification, Converting Ideas, and Providing Information types. Latent Dirichlet allocation (LDA) will not work because LDA heavily relies on the similarity of the used terminology in two files. In the aforementioned types, the used terminology across two files tends to be different. Other techniques, such as clone detection techniques cannot capture the Non-cosmetic Modification, Converting Ideas, and Providing Information types either. Hence, we had no option other than to perform our analysis in a manual fashion. To mitigate the threat of bias during the manual analysis, two of the research collaborators conducted the manual analysis and discussed any conflicts with a third research collaborator until a consensus was reached. We used Cohen's kappa [42] to measure the inter-rater agreement. The kappa values ranged from 0.79 to 0.92, which implies a high level of agreement. In addition, to improve the replicability of our study, we made our studied data and results (including the coding results of our exploratory study and survey) available in our online appendix [111].

Another threat is that our findings in RQ2 may not exactly reflect the intent of a user who reused code from Stack Overflow, since the code provided in the post to which the link points may not exactly be what the developer needs. To mitigate this threat, we tried contacting the developers about their intent, but received no response.

Construct validity. A threat to the construct validity of this study is that we used mostly closed-ended questions in our survey, which may affect the richness of the responses collected from participants. However, open-ended questions have several disadvantages [71]: (1) Open-ended questions take

much longer for participants to fill out, making it more likely that they do not fill out the survey at all. (2) Open-ended questions have the problem of missing data, i.e., participants skipping or doing a poor job at answering a question (i.e., by giving an incomplete or invalid answer). (3) Open-ended questions are much more difficult to code than closed-ended questions. We felt that for most of our survey, the advantages of closed-ended questions outweighed the disadvantages. Therefore, we chose to use mostly closed-ended questions, together with an "Other." field. Another threat is that Question 18 (see the appendix for details) could influence the participants' answer to Question 19. We used Question 18 as an "icebreaker", so that the participants could start their thought process from there. Future studies should consider this effect when performing user surveys.

3.8 Conclusion of This Chapter

Prior studies on code reuse from Q&A platforms have focused on locating and integrating source code in a required context automatically. However, no studies have been done on *how* developers utilize source code from Q&A platforms. In this chapter, we studied how developers reuse code from Q&A platforms and we identified the barriers that they faced during the code reuse process. The most important findings of our study are:

1. Our exploratory study shows that 78.2% of the reused source code from Stack Overflow had to be modified in order to work in the required context. The required modification ranged from simple refactoring to a complete reimplementation.

2. Developers reuse source code snippets from non-accepted answers as well (26% of the studied cases).

3. Developers prefer reimplementing source code over reusing source code because of the difficulty of integrating the code into their own project, and low comprehension or low quality of the code.

4. The most suggested improvements for next-generation Q&A platforms are about improving the quality of source code snippets.

5. Many developers do not understand, or do not seem to care much about the code license of Q&A platforms.

Our study shows that Q&A platforms are evolving beyond their traditional use of asking and answering questions. From our survey, we can conclude that code reuse from Q&A platforms is a real challenge for developers: judging by the suggestions that we extracted from the survey,

many developers face similar barriers during the code reuse process. Next-generation Q&A platforms should integrate existing solutions that improve the quality, comprehension and organization of source code snippets to better facilitate code reuse. Researchers can leverage the roadmap that is presented in this chapter to remove many of these barriers in code reuse.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This dissertation presented our studies on the two aspects of code reuse: software license and source code.

Firstly, we conducted an empirical study on Debian 7.5 and a collection of Java projects to analyze the issue of license inconsistencies. From the result, we generalized 3 types of license inconsistencies: *i*) license addition or removal *ii*) license upgrade or downgrade, and *iii*) license change. Meanwhile, we investigated the causes of these license inconsistencies and generalized them into 4 reasons: *i*) author's modification *ii*) resolving a multi-license *iii*) reuser's addition to original license, and *iv*) reuser's modification. Among these 4 categories, we consider the last two are potentially risky which needs further attention. This finding supports our assumption that license inconsistency brings potential license violation issues. Thus we call for the attention from developers that they should be careful about the license inconsistency issues in their own projects. We think that our proposed method can be a good start point to find out those license inconsistency cases. Having a clear clue of the license inconsistency issues would help the developers/organization avoid potential legal disputes.

Secondly, we investigated the source code change during code reuse. With a manual analysis on how developers integrate source code from Stack Overflow into their own projects, we found that it is not uncommon that developers make modifications to the source code they reuse. The modifications vary from simple refactoring to a fully rewriting of the algorithm. These required modifications may indicate potential barriers to the developers who performs code reuse. With this in mind, we designed a survey on 453 developers, which aims to find out what developers are suffering during the reuse process and provide useful insights for a next-generation Q&A platforms. We analyzed the responses from the participants and generalized 4 main reasons that hinder code reuse: *i*) code need to be modified

according to the context *ii*) developers do not comprehend the source code to be reused *iii*) low quality of the source code, and *iv*) reusing source is time consuming. Further more, we summarized a roadmap for a next-generation Q&A platforms which is generation in 5 categories: (1) code quality, (2) information enhancement & management, (3) data organization, (4) license, and (5) human factor. We believe that these findings give directions to future research that aim to improve code reuse. Practitioners can use the roadmap as a guideline to build a next-generation Q&A platforms.

Overall, this dissertation provides an insight into the legal aspect (i.e., software license) and the core part (i.e., source code) of code reuse. We believe that the findings in this dissertation point out some future directions for researchers who are to improve code reuse and provide some guidelines for practitioners to create a better platform for code reuse.

4.2 Future Directions

In our first study of license inconsistencies, we discovered the difficulty of determining the provenance of a source code file. This difficulty prevents us from distinguish the direction of code reuse, i.e, we do not know whether the source code in file A was reused from file B, or the other way around. In other words, when license inconsistency exists, it is difficult to determine who is to blame for its occurrence. Future research could be conducted to solve this issue, thus helping narrow down the truly risky license inconsistency cases and their causes.

Our second study of the code reuse from Q&A platforms suggests several future directions to improve code reuse. For example, studies on improving the code quality on Q&A platforms would remove the biggest concerns from the developers; studies on enhancing information and data organization of Q&A platforms would help creating a platform with richer information (e.g., API documentation) that could be easily accessed. We believe that these enhancements would help developers reuse source code from Q&A platforms more effectively and efficiently.

Bibliography

[1] Abdalkareem, R., Shihab, E., and Rilling, J. (2017). What do developers use the crowd for? a study using Stack Overflow. *IEEE Software*, **34**(2), 53–60.

[2] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., and Schneider, K. A. (2016). Mining duplicate questions in Stack Overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR)*, pages 402–412.

[3] Almeida, D. A., Murphy, G. C., Wilson, G., and Hoye, M. (2017). Do software developers understand open source licenses? In *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, pages 1–11. IEEE.

[4] Alnusair, A., Rawashdeh, M., Hossain, M. A., and Alhamid, M. F. (2016). Utilizing semantic techniques for automatic code reuse in software repositories. In *Quality Software Through Reuse and Integration*, pages 42–62. Springer.

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

[6] An, L., Mlouki, O., Khomh, F., and Antoniol, G. (2017). Stack Overflow: A code laundering platform? In *Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 283–293. IEEE.

[7] Anderson, A., Huttenlocher, D., Kleinberg, J., and Leskovec, J. (2013). Steering user behavior with badges. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 95–106. ACM.

[8] Apte, U., Sankar, C. S., Thakur, M., and Turner, J. E. (1990). Reusability-Based Strategy for Development of Information Systems: Implementation Experience of a Bank. Technical Report 4.

[9] Armaly, A. and McMillan, C. (2016). Pragmatic source code reuse via execution record and replay. *Journal of Software: Evolution and Process*, **28**(8), 642–664.

[10] Atwood, J. (2009). Attribution required – Stack Overflow blog. `https://stackoverflow.blog/2009/06/25/attribution-required/`. (last visited: Aug 25, 2017).

[11] Azad, S., Rigby, P. C., and Guerrouj, L. (2017). Generating API call rules from version history and stack overflow posts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **25**(4), 29.

[12] Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., and Lopes, C. (2006). Sourcerer: A search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 681–682. ACM.

[13] Barzilay, O. (2011). Example embedding. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 137–144.

[14] Basili, V. R., Briand, L. C., and Melo, W. L. (1996). How reuse influences productivity in object-oriented systems. *Communications of the ACM*, **39**(10), 104–116.

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

[16] Bian, J., Gao, B., and Liu, T.-Y. (2014). *Knowledge-Powered Deep Learning for Word Embedding*, pages 132–148. Springer Berlin Heidelberg.

[17] Boehm, B. W. (1987). Improving software productivity. *Computer*, **20**(9), 43–57.

[18] Cavusoglu, H., Li, Z., and Huang, K.-W. (2015). Can gamification motivate voluntary contributions?: The case of StackOverflow Q&A community. In *Proceedings of the 18th ACM Conference Companion on Computer Supported Cooperative Work & Social Computing*, pages 171–174. ACM.

[19] Chen, C., Gao, S., and Xing, Z. (2016). Mining analogical libraries in Q&A discussions - incorporating relational and categorical knowledge

into word embedding. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 338–348. IEEE.

[20] Chen, C., Xing, Z., and Wang, X. (2017). Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*, pages 450–461. IEEE.

[21] Cohen, J. (1968). Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, **70**(4), 213.

[22] Cottrell, R., Walker, R. J., and Denzinger, J. (2008). Semi-automating small-scale source code reuse via structural correspondence. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT)*, pages 214–225. ACM.

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

[24] Feldthaus, A. and Møller, A. (2013). Semi-automatic rename refactoring for javascript. In *Proceedings of the 2013 ACM SIGPLAN International Conference On Object Oriented Programming Systems Languages & Applications*, volume 48, pages 323–338. ACM.

[25] Foundation, F. S. (2006). License violations and compliance. `https://www.fsf.org/licensing/compliance`. (Accessed on 01/07/2019).

[26] Free Software Foundation, I. (2007). The gnu general public license v3.0. `http://www.gnu.org/licenses/gpl.html`. (Accessed on 12/14/2018).

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

[28] Galenson, J., Reames, P., Bodik, R., Hartmann, B., and Sen, K. (2014). Codehint: Dynamic and interactive synthesis of code snippets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 653–663.

[29] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[30] Ganguly, D., Roy, D., Mitra, M., and Jones, G. J. (2015). Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 795–798.

[31] Gao, Q., Zhang, H., Wang, J., Xiong, Y., Zhang, L., and Mei, H. (2015). Fixing recurring crash bugs via analyzing Q&A sites. In *Proceedings of the 30th International Conference on Automated Software Engineering (ASE)*, pages 307–318.

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

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

[34] German, D. M. and Hassan, A. E. (2009). License integration patterns: Addressing license mismatches in component-based development. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 188–198. IEEE.

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

[36] Gharehyazie, M., Ray, B., and Filkov, V. (2017). Some from here, some from there: Cross-project code reuse in github. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 291–301. IEEE.

[37] Glaser, B. (2017). *Discovery of grounded theory: Strategies for qualitative research*. Routledge.

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

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

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

[41] Gu, X., Zhang, H., Zhang, D., and Kim, S. (2016). Deep API learning. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 631–642. ACM.

[42] Gwet, K. *et al.* (2002). Inter-rater reliability: dependency on trait prevalence and marginal homogeneity. *Statistical Methods for Inter-Rater Reliability Assessment Series*, **2**, 1–9.

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

[44] Hua, L., Kim, M., and McKinley, K. S. (2015). Does automated refactoring obviate systematic editing? In *IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, volume 1, pages 392–402. IEEE.

[45] Initiative, O. S. (????). The 2-clause bsd license. `https://opensource.org/licenses/BSD-2-Clause`. (Accessed on 12/14/2018).

[46] Jones, T. C. (1984). Reusability in Programming: A Survey of the State of the Art. *IEEE Transactions on Software Engineering*, **SE-10**(5), 488–494.

[47] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 92–101. ACM.

[48] Kamiya, T., Kusumoto, S., and Inoue, K. (2002). CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, **28**(7), 654–670.

[49] Knight, J. C. and Dunn, M. F. (1998). Software quality through domain-driven certification. *Annals of Software Engineering*, **5**(1), 293.

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

[51] Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, **24**(2), 131–183.

[52] Krumia (2014). Introduce an “obsolete answer" vote. `https://meta.stackoverflow.com/questions/272651/introduce-an-obsolete-answer-vote`. (last visited: Aug 25, 2017).

[53] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2267–2273. AAAI Press.

[54] Latoza, T. T. D. and van der Hoek, A. (2016). Crowdsourcing in Software Engineering : Models , Opportunities , and Challenges. Technical report.

[55] Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O., and Morisio, M. (2009). Development with off-the-shelf components: 10 facts. *IEEE Software*, **26**(2), 80–87.

[56] Liu, P., Joty, S. R., and Meng, H. M. (2015). Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1433–1443. The Association for Computational Linguistics.

[57] Lv, F., Zhang, H., Lou, J.-g., Wang, S., Zhang, D., and Zhao, J. (2015). CodeHow: Effective code search based on API understanding and extended boolean model. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 260–270. IEEE.

[58] Madison, M. J. (2003). Reconstructing the software license. *Loy. U. Chi. Lj*, **35**, 275.

[59] Manabe, Y., Hayase, Y., and Inoue, K. (2010). Evolutional analysis of licenses in FOSS. In *Proceedings of the Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (IWPSE-EVOL2010)*, pages 83–87.

[60] Manabe, Y., German, D., and Inoue, K. (2014). Analyzing the relationship between the license of packages and their files in free and open source software. In *Proceedings of the 10th International Conference on Open Source Systems (OSS2014)*, pages 51–60.

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

[62] McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C. (2011). Portfolio: Finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 111–120.

[63] Meng, N., Kim, M., and McKinley, K. S. (2011). Systematic editing: Generating program transformations from an example. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 329–342.

[64] Meng, N., Kim, M., and McKinley, K. S. (2013). Lase: locating and applying systematic edits by learning from examples. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 502–511. IEEE.

[65] Nguyen, A. T., Nguyen, T. T., Nguyen, H. A., Tamrawi, A., Nguyen, H. V., Al-Kofahi, J., and Nguyen, T. N. (2012). Graph-based pattern-oriented, context-sensitive source code completion. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 69–79.

[66] Ponzanelli, L., Bacchelli, A., and Lanza, M. (2013). Leveraging crowd knowledge for software comprehension and development. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 57–66. IEEE.

[67] Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., and Lanza, M. (2014a). Mining stackoverflow to turn the IDE into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 102–111. ACM.

[68] Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., and Lanza, M. (2014b). Prompter: A self-confident recommender system. In *ICSME*, pages 577–580.

[69] Ponzanelli, L., Mocci, A., Bacchelli, A., and Lanza, M. (2014c). Understanding and classifying the quality of technical forum questions. In *Proceedings of the 14th International Conference on Quality Software (QSIC)*, pages 343–352.

[70] Raychev, V., Vechev, M., and Yahav, E. (2014). Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 419–428.

[71] Reja, U., Manfreda, K. L., Hlebec, V., and Vehovar, V. (2003). Open-ended vs. close-ended questions in web questionnaires. *Developments in Applied Statistics (Metodološki zvezki)*, **19**, 159–77.

[72] Rigby, P. C. and Robillard, M. P. (2013). Discovering essential code elements in informal documentation. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, pages 832–841. IEEE.

[73] Roy, C. K., Cordy, J. R., and Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, **74**(7), 470–495.

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

[75] Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering (TSE)*, **25**(4), 557–572.

[76] Seaman, C. B., Shull, F., Regardie, M., Elbert, D., Feldmann, R. L., Guo, Y., and Godfrey, S. (2008). Defect categorization: making use of a decade of widely varying historical data. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 149–157. ACM.

[77] Searchcode (2016a). searchcode - API. `https://searchcode.com/api/`. (last visited: Aug 25, 2017).

[78] Searchcode (2016b). searchcode - Homepage. `https://searchcode.com/`. (last visited: Aug 25, 2017).

[79] Sillito, J., Maurer, F., Nasehi, S. M., and Burns, C. (2012). What makes a good code example?: A study of programming Q&A in StackOverflow. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34.

[80] Singh, P. V., Tepper, D. A., and Phelps, C. (2012). Networks, Social Influence, and the Choice Among Competing Innovations: Insights from Open Source Software Licenses. pages 1–22.

[81] Sojer, M., Henkel, J., Wade, M., and Crowston, K. (2010). Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. Technical report.

[82] Stack Exchange (2015). The MIT license — clarity on using code on Stack Overflow and Stack Exchange. `https://meta.stackexchange.com/q/271080/337948`. (last visited: Aug 25, 2017).

[83] Stack Exchange (2017). All sites - Stack Exchange. https://stackexchange.com/sites. (last visited: Aug 25, 2017).

[84] Stack Overflow (2014). Feedback requested: Runnable code snippets in questions and answers. https://meta.stackoverflow.com/questions/269753/feedback-requested-runnable-code-snippets-in-questions-and-answers. (last visited: Aug 25, 2017).

[85] Stack Overflow (2016). Stack Overflow developer survey results 2016. http://stackoverflow.com/research/developer-survey-2016. (last visited: Aug 25, 2017).

[86] Stack Overflow (2017). Stack Overflow - Homepage. https://stackoverflow.com/. (last visited: Aug 25, 2017).

[87] Standish, T. A. (1984). An essay on software reuse. *IEEE Transactions on Software Engineering*, **SE-10**(5), 494–497.

[88] Tan, G. (2016). A collection of well-known software failures. http://www.cse.psu.edu/~gxt29/bug/softwarebug.html. (Accessed on 12/05/2018).

[89] Treude, C. and Robillard, M. P. (2016). Augmenting API documentation with insights from Stack Overflow. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 392–403. ACM.

[90] Treude, C. and Robillard, M. P. (2017). Understanding stack overflow code fragments. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*, pages 509–513.

[91] Treude, C., Barzilay, O., and Storey, M.-A. (2011). How do programmers ask and answer questions on the web? (NIER track). In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 804–807.

[92] Tuunanen, T., Koskinen, J., and Kärkkäinen, T. (2009). Automated software license analysis. *Automated Software Engineering*, **16**(3-4), 455–490.

[93] Vasilescu, B., Filkov, V., and Serebrenik, A. (2013). StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In *Proceedings of 2013 International Conference on Social Computing (SocialCom)*, pages 188–195. IEEE.

[94] Vassallo, C., Panichella, S., Di Penta, M., and Canfora, G. (2014). CODES: Mining source code descriptions from developers discussions. In *Proceedings of the 22nd International Conference on Program Comprehension (ICPC)*, pages 106–109.

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

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

[97] Wang, H., Lu, Y., and Zhai, C. (2010). Latent aspect rating analysis on review text data: A rating regression approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 783–792.

[98] Wang, S., Lo, D., and Jiang, L. (2014a). Active code search: Incorporating user feedback to improve code search relevance. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 677–682.

[99] Wang, S., Lo, D., Vasilescu, B., and Serebrenik, A. (2014b). EnTagRec: An enhanced tag recommendation system for software information sites. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 291–300.

[100] Wang, S., Lo, D., and Jiang, L. (2016a). Autoquery: automatic construction of dependency queries for code search. *Automated Software Engineering*, **23**(3), 393–425.

[101] Wang, S., Lo, D., Vasilescu, B., and Serebrenik, A. (2017a). EnTagRec ++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering*.

[102] Wang, S., Chen, T.-H., and Hassan, A. E. (2017b). Understanding the factors for fast answers in technical Q&A websites. *Empirical Software Engineering*, pages 1–42.

[103] Wang, X., Pollock, L. L., and Vijay-Shanker, K. (2014c). Automatic segmentation of method code into meaningful blocks: Design and evaluation. *Journal of Software: Evolution and Process*, **26**(1), 27–49.

[104] Wang, X., Pollock, L. L., and Vijay-Shanker, K. (2017c). Automatically generating natural language descriptions for object-related statement sequences. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 205–216.

[105] Wang, Y., Feng, Y., Martins, R., Kaushik, A., Dillig, I., and Reiss, S. P. (2016b). Hunter: next-generation code reuse for java. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 1028–1032. ACM.

[106] Wang, Z., Hamza, W., and Florian, R. (2017d). Bilateral multi-perspective matching for natural language sentences. *CoRR*, **abs/1702.03814**.

[107] Wikipedia (????). Oracle america, inc. v. google, inc. `https://en.wikipedia.org/wiki/Oracle_America,_Inc._v._Google,_Inc.?oldformat=true`. (Accessed on 01/07/2019).

[108] Wong, E., Yang, J., and Tan, L. (2013). Autocomment: Mining question and answer sites for automatic comment generation. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 562–567. IEEE.

[109] Wong, T.-L., Lam, W., and Wong, T.-S. (2008). An unsupervised framework for extracting and normalizing product attributes from multiple web sites. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 35–42.

[110] Wu, Y., Manabe, Y., Kanda, T., German, D. M., and Inoue, K. (2015). A method to detect license inconsistencies in large-scale open source projects. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR2015)*, pages 324–333.

[111] Wu, Y., Wang, S., Bezemer, C.-P., and Inoue, K. (2017). Online appendix of manuscript "How Do Developers Utilize Source Code from Stack Overflow?". `https://zenodo.org/record/1116508`.

[112] Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., and Xing, Z. (2017). What do developers search for on the web? *Empirical Software Engineering*.

[113] Xin, X., Lingfeng, B., David, L., Zhenchang, X., Ahmed, E. H., and Shanping, L. (2017). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering (TSE)*, **99**(26).

[114] Yellin, D. M. and Strom, R. E. (1997). Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, **19**(2), 292–333.

[115] Yin, P. and Neubig, G. (2017). A syntactic neural model for general-purpose code generation. *CoRR*, **abs/1704.01696**.

[116] Yu, J., Zha, Z.-J., Wang, M., and Chua, T.-S. (2011). Aspect ranking: Identifying important product aspects from online consumer reviews. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, pages 1496–1505.

[117] Zagalsky, A., German, D. M., Storey, M.-A., Teshima, C. G., and Poo-Caamaño, G. (2017). How the R community creates and curates knowledge: an extended study of Stack Overflow and mailing lists. *Empirical Software Engineering*.

[118] Zhang, H., Shi, B., and Zhan, L. (2010). Automatic checking of license compliance. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–3. IEEE.

[119] Zhang, W. E., Sheng, Q. Z., Lau, J. H., and Abebe, E. (2017). Detecting duplicate posts in programming qa communities via latent semantics and association rules. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 1221–1229.

[120] Zhang, Y., Lo, D., Xia, X., and Sun, J.-L. (2015). Multi-factor duplicate question detection in Stack Overflow. *Journal of Computer Science and Technology*, **30**(5), 981–997.

[121] Zhao, L. and Li, C. (2009). *Ontology Based Opinion Mining for Movie Reviews*, pages 204–214. Springer Berlin Heidelberg, Berlin, Heidelberg.

[122] Zhou, P., Liu, J., Yang, Z., and Zhou, G. (2017). Scalable tag recommendation for software information sites. In *Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 272–282. IEEE.

Appendix

Below are the questions and options in our online survey. Single-selection options are marked with circle marks (○) in front; multi-selection options are marked with box marks (□) in front. When participants choose the option *"Other,"* they are allowed to input a free text as an additional answer.

Part I			
1. How many years of software engineering experience do you have?			
○ 1	○ 2	○ 3	○ 4
○ 5	○ 6	○ 7	○ 8
○ 9	○ 10+		
2. What type of project(s) are you working on?			
□ Open source	□ Personal	□ Industrial	□ Academic
□ Other			
3. Which programming language(s) do you use in your projects?			
□ Java	□ C/C++	□ C#	□ Python
□ Visual Basic .Net	□ JavaScript	□ Assembly	□ PHP
□ Perl	□ Ruby	□ Other	
4. How often do you use Q&A platforms?			
○ Every day	○ Once a week	○ Once a month	
○ Once every few months or less		○ Never	
5. What do you use Q&A platforms for?			
□ Learning new techniques/methodologies			
□ Refreshing the knowledge of old techniques/methodologies			
□ Solving a specific programming issue			
□ Finding references that I can refer to in my source code to make future maintenance easier			
□ Answering questions			
□ Other			
6. Which Q&A platforms do you use to look for solutions to programming-related issues?			
□ Stack Overflow			
□ Quora			
□ Product-specific support forums			

□ Language-specific support forums
□ I do not use Q&A platforms for this purpose
□ Other
7. Have you ever reused source code from a Q&A platform?
○ Yes ○ No

Part II

8. How often do you *reuse* source code from Q&A platforms?
○ Every day ○ Once a week ○ Once a month
○ Once every few months or less ○ Never
9. How often do you *reimplement* source code from Q&A platforms?
○ Every day ○ Once a week ○ Once a month
○ Once every few months or less ○ Never
10. If you prefer reimplementing the source code over reusing existing code, why?
□ Code should be written in relation to the context.
□ I don't want to reuse source code that I don't fully comprehend.
□ The quality of the existing source code is too low.
□ Re-implementing the source code takes less time than reusing.
□ Other
11. What do you consider the most important factors when deciding when to reuse code from a Q&A platform?
□ Correctness (i.e., bug-free)
□ Performance (i.e., efficient)
□ Readability (i.e., easy to read/understand)
□ Simplicity (i.e., less lines of code)
□ Compatibility (e.g, support more platforms)
□ Whether the answer is accepted by the questioner.
□ Whether the answer has the highest number of upvotes.
□ Other
12. Which aspects cause you difficulty when reusing source code from Q&A platforms?
□ Syntax errors need to be fixed to make the source code runnable.
□ Bugs (e.g., index out of bounds) need to be fixed.
□ Readability needs to be improved.
□ Performance needs to be improved.
□ The code snippet is not in the programming language I need.
□ Code needs to be adapted to my specific use case.
□ The license terms of the Q&A platform are unclear.
□ Other
13. Do you always refer to the Q&A platform post from which you reused source code in your documentation or code comments? Why (not)?
○ Yes, I add a link to the post/answer to show my respects/appreciation to the original author.

◦ Yes, because it is required by the license terms of that Q&A platform (e.g., CC-BY-SA 3.0 in the case of Stack Overflow).
◦ Yes, to make it easier for future maintenance.
◦ No, I would like to, but always forget.
◦ No, I don't do that. (Please elaborate the reason below if you could)
◦ Other

14. Are you aware of the license terms of reused source code from a Q&A platform?
◦ Yes, I fully understand the license terms.
◦ Yes, I know about the existence of such terms, but I am not sure what obligations I have.
◦ No, I did not know about them, but I would like to learn more about them.
◦ No, I did not know about them, neither do I care about them.
◦ Other

15. Which license(s) do the projects into which you reused code from a Q&A platform use?
□ GPL family (any version of LGPL, GPL or AGPL) □ MIT License
□ Apache License □ BSD License
□ A proprietary license □ No license
□ I don't know □ Other

16. In general, would you say that the license(s) of these project(s) are compatible with the license of the Q&A platform from which you reused source code?
◦ Strongly disagree ◦ Disagree ◦ Neutral
◦ Strongly agree ◦ Agree

17. How important is it to have more detailed information about the license terms and legal obligations of reusing source code from Q&A platforms?
◦ Very unimportant ◦ Unimportant ◦ Neutral
◦ Very important ◦ Important

Part III

18. How useful would it be to let other users tag answers on Q&A platforms with labels that describe the source code in an answer as 'performant', 'correct', 'readable', etc.?
◦ Very unuseful ◦ Unuseful ◦ Neutral
◦ Very useful ◦ Useful

19. If you could give any suggestions (regardless of whether they are feasible) for a next-generation code Q&A platform, what would you suggest?