The Journal of Systems and Software 210 (2024) 111964

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

An empirical study of code reuse between GitHub and stack overflow during software development☆

Xiangping Chen [a], Furen Xu [b], Yuan Huang [b,*], Xiaocong Zhou [c], Zibin Zheng [b]

[a] Sun Yat-sen University and School of Journalism and Communication, China
[b] Sun Yat-sen University and School of Software Engineering, China
[c] Sun Yat-sen University and School of Computer Science and Engineering, China

ARTICLE INFO

Keywords:
Stack overflow
Code reuse
Code clone
Semantic analysis

ABSTRACT

With the rise of programming Q&A websites (e.g., Stack Overflow) and the open-source movement, code reuse has become a common phenomenon. Our study aims to provide a comprehensive study of the code reuse behavior of programmers during software development, i.e., we mainly focus on the code reuse between the code snippets in the commits of open-source projects and the code snippets on Stack Overflow (SO). The open-source java project code dataset we construct contains 793 projects which include 342,148 modified code snippets, and the SO code dataset includes 1,355,617 posts. Then, we employ a code clone detection tool to identify the instances of code reuse between the modified code snippets of commits and the code snippets of the SO posts. We find that the average code reuse ratio of the projects is 6.32%, which will have a significant upward trend in the future. Additionally, we find that experienced developers seem to be more likely to reuse the code on SO, and prefer to reuse posts with more favorites and higher scores. We combine deep learning and topic analysis algorithms to fully exploit the semantic information of SO posts. The result shows a certain difference in the distribution of post types reused by bug-related commits and non-bug-related commits. We also discover that the code reuse ratio (14.44%) in java class files that have undergone multiple modifications is more than double the overall code reuse ratio (6.32%). Finally, we discuss the reuse habits of programmers and find that they may refer to multiple posts in one reuse, and these posts are related to a certain extent. From these results, our study provides multiple practical insights for different stakeholders: researchers, developers, and the SO platform.

1. Introduction

In software development, programmers often reuse code from Q&A websites to solve programming problems they encounter (Vasilescu et al., 2013; Gallardo-Valencia and Sim, 2009; Brandt et al., 2009). Q&A websites that have emerged in recent years, such as the most well-known Stack Overflow (SO), have attracted the attention of many programmers and researchers. Users can not only find the possible solutions from SO in a short time but also help others to solve problems (Ahmad et al., 2019; Zhang et al., 2019b; Yang et al., 2016). Previous studies show that users query programming-related issues on the Q&A websites and receive a response within an average of 11 min (Mamykina et al., 2011), and a case study from Google shows that programmers issue an average of 12 code search queries per working day (Sadowski et al., 2015).

The data from SO shows that the SO community is quite active, with 1,581,814 posts about the Java programming language, and roughly 392 posts per day. It is evident that SO has provided many programmers with a great deal of assistance and convenience. Code reuse is formally defined as "using existing software or software knowledge to build new software" (Frakes and Kang, 2005). According to a prior study (Fischer et al., 2017), the answers of a majority of SO posts contain code snippets and explanations (Yang et al., 2016), many programmers reuse code snippets on SO when they find answers

☆ Editor: Hongyu Zhang.

* Corresponding author.
E-mail addresses: chenxp8@mail.sysu.edu.cn (X. Chen), xufr@mail2.sysu.edu.cn (F. Xu), huangyuan5@mail.sysu.edu.cn (Y. Huang), isszxc@mail.sysu.edu.cn (X. Zhou), zhzibin@mail.sysu.edu.cn (Z. Zheng).

[1] https://searchcode.com/codesearch/view/43953373/.

https://doi.org/10.1016/j.jss.2024.111964
Received 12 July 2023; Received in revised form 20 December 2023; Accepted 3 January 2024
Available online 8 January 2024

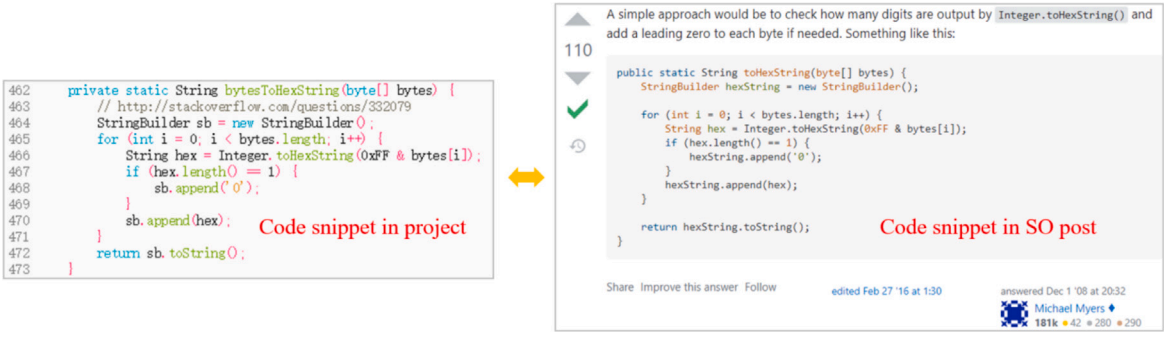

Fig. 1. Code reuse example between SO and project.

that have functional matches to their programming questions from the SO platform. Reusing code from SO can bring great convenience to programmers, which not only improves their work efficiency but also improves their programming skills. Fig. 1 depicts a genuine example[1] of how a programmer would reuse SO code when creating a program function, i.e., `checking the number of digits returned by Integer.toHexString() and, if necessary, adding a leading zero to each byte`.

In our study, we analyze the code reuse between the code snippets involved in commits from GitHub and the code snippets from SO (i.e. code reuse analysis in the evolution of software projects) to discover how programmers reuse code on SO. The commits from a project record the continuous development process including the bug fixing and the new features. By examining the code reuse between the modified code snippets in the commits and the code snippets on SO, we can assess code reuse during development. Static source code refers to the complete compilable code, while commit (i.e., the change code) refers to the code snippet that is modified or updated by the programmer. When programmers modify or update the code during the development process, they use the 'git commit' command to submit the modified or updated code. So studying commits means the study focuses on reuse cases during software development. Studying commits is important and essentially different. The commit phase is an early and important stage in the software development cycle, different studies on commit have been carried out in many fields, such as research on just-in-time defect prediction and localization (Ni et al., 2022; Hoang et al., 2020), they use machine learning methods to model and identify whether the defects are introducing by commits. These defects may also be introduced by programmers reusing code. Additionally, some projects have long maintenance cycles and fast software development iterations. Therefore, studying commits is important. As a result, we start by crawling the 793 most popular java projects on GitHub, which include 342,148 modified code snippets. We filter the posts from SO that contain code snippets and get 1,355,617 posts that are related to java.

To examine the code reuse activity from SO during software development, we need to determine the reused pairs of the code snippets between SO and GitHub. Specifically, we use a token-based code cloning detecting technique named CCFinder, which is developed by Kamiya et al. (2002). We leverage CCFinder to identify the code cloning relationship of the code snippets between the SO and GitHub. We take advantage of the code clone detection results and introduce the time constraint to determine the code reused pairs, which is similar to the method utilized in these studies (Abdalkareem et al., 2017; An et al., 2017). We did not use the explicit links between SO and GitHub (i.e., some code in GitHub reused from SO has an explicit link, such as the case in https://github.com/huangyuan6/jquery-handsontable/blob/master/demo/js/samples.js#L51) to determine the code reuse, because there is a limited proportion of code reuse recorded SO links (Wu et al., 2019). We count on the 793 projects with 98,283 commits, of which only 26 commits have explicit SO links, which is a small proportion, less than 3/10,000. We build a large-scale dataset because we aim at exploring the general state of the code reuse activity. Therefore, using explicit links is inappropriate.

We propose the following research questions:

RQ1. How active is code reuse in development?

RQ2. What is the relationship between programmer experience and reuse ratio?

RQ3. Are code snippets reused in bug-related changes more likely to have been reused from SO?

RQ4. Are reused code snippets present in commits more likely to undergo repeated modifications?

RQ5. How do programmers behave when they reuse code?

We come to some important conclusions regarding the code reuse between the code snippets in the commits of open-source projects and the code snippets on SO, which can provide multiple practical insights for different stakeholders: researchers, developers, SO platform.

Replication Package. The replication package and the dataset are available at https://github.com/love-SE/Code-Reuse-Analysis.

Paper Organization. Section 2 describes how we construct the dataset. Section 3 shows our methodology in detail. Section 4 presents the research questions and analyzes the survey results. Section 5 describes the related work of the study. Section 6 introduce the threats to validity. Section 7 concludes our study.

2. Data collection

2.1. Building Stack Overflow code dataset

We download the posts (a ".xml" format file) as the SO data source from StackExchange which is the public data dump source of the website. The SO posts in the file cover the period from August 1, 2008 (the establishment time of SO), to February 28, 2021. According to Fig. 3, each SO post includes specific information including the question title, release date, and question tag, the score, the favorite count, the comment count, and the answer count (see Fig. 2).

To obtain the data we require, we secondly parse the XML formatted post data. We specifically choose posts with tags of '<java>'. After filtering, there are 1,581,814 posts with java-related content in total. The tag mechanism of Stack Overflow is: those tags that are closely related to Java, such as "jdk", "jre", "javax" will be remapped to the "java" tag on Stack Overflow (shown in Fig. 4). Therefore, we only need to use the question tag "java" to collect the SO code dataset.

Further, since we aim at exploring the code reuse activity, we filter out SO posts without code snippets based on the tag "<code><\code>" in the posts. Considering that the detection method may be hard to find out code clone accurately if the code snippet is very short. For example, it may identify common statements like single-line "while statements" or "if statements" as code clones. Therefore, we filter out code snippets that are too short in advance in the data filtering stage. The total number of posts is 1,355,617 after removing the code snippet whose token length is less than 25 (the threshold selection will be covered in

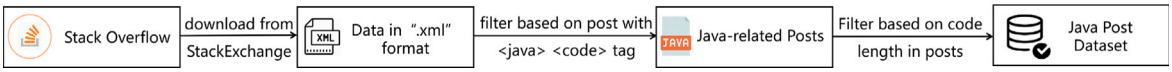

Fig. 2. Collection process of Stack Overflow posts.

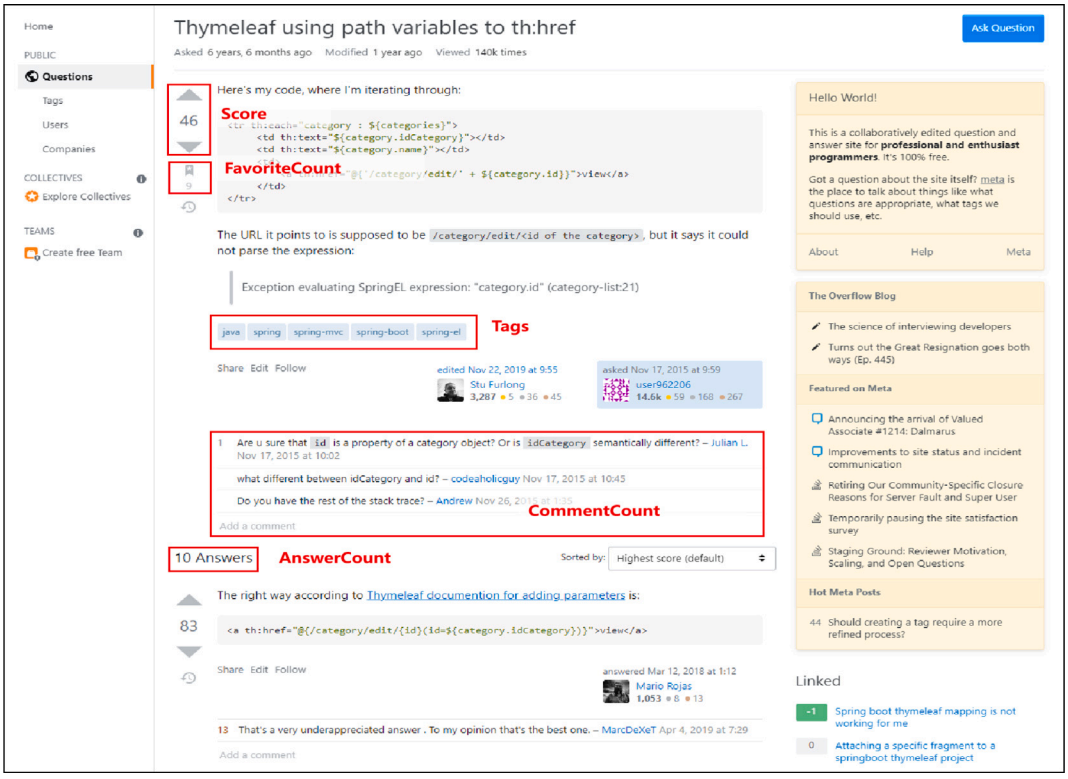

Fig. 3. Information of the SO post.

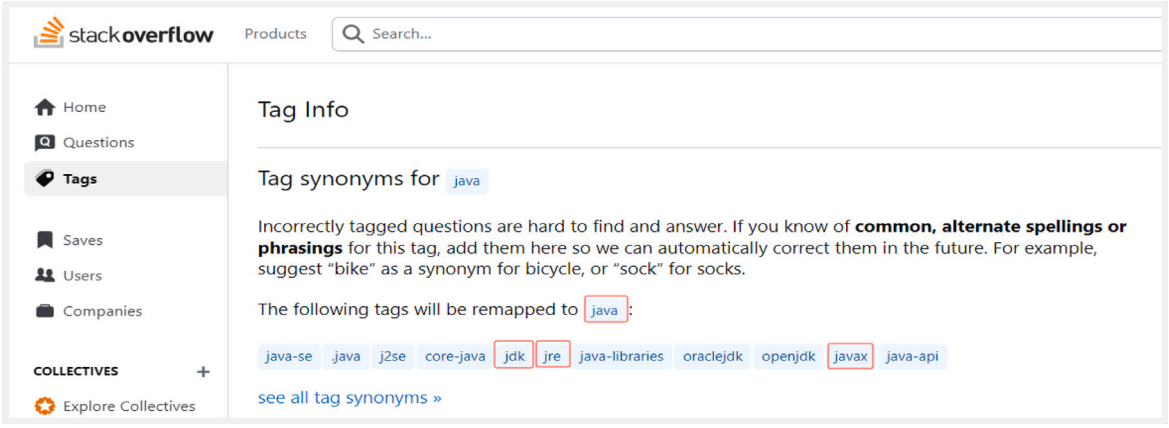

Fig. 4. Tag synonyms for "java" on Stack Overflow.

Section 3). The quantity of post-data following each filtering stage is shown in Table 1.

We use the 1,355,617 posts as the source for constructing the SO code dataset. Each SO post includes the original text data and code snippets. We extract the creation date data ("CreationDate"), title data ("Title"), tag data ("Tags"), the score, the favorite count, the comment count, and the answer count. (Score is the post information we take from the question of the post, because when programmers search for answers on Stack Overflow, they can quickly retrieve high-quality answers by giving search results in descending order of scores according to the search conditions.)

Table 1
Number of SO posts.

Each stage	Posts count
All posts	45,919,817
Posts with '<java>' tags	1,581,814
Posts filtered by length	1,355,617

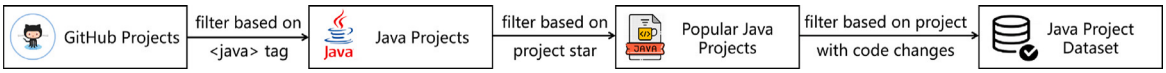

Fig. 5. Collection process of GitHub project.

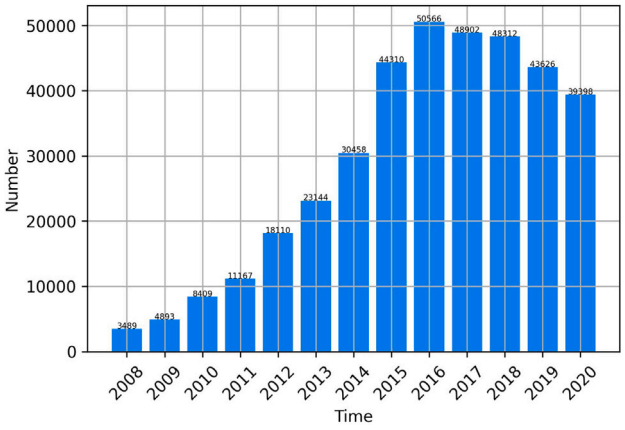

Fig. 6. Distribution of the commit time.

2.2. Building open-source project code dataset

First of all, we crawl the most popular 1000 popular Java projects on GitHub based on the project stars and project tag ("<java>" tag) (see Fig. 5).

The commit time of these projects spans the period from 2001/1/3 to 2020/12/14. At the same time, we filter out some projects that belong to the study notes category (such as the project "toBeTop-Javaer"). The commits in such projects are usually the submission and modification of some texts, which do not reflect the code changes in the project development process. After filtering, we obtain a total of 793 popular java projects and their related commit records.

Secondly, since we want to explore the reuse of SO knowledge in the process of project code evolution, we need to extract modified code snippets according to the project's commit data. To extract the code snippets of the commits, we adopt the Change Distilling algorithm (Fluri et al., 2007). Through comparing the abstract syntax trees of the new and old code java files included in a commit for extracting fine-grained code changes.

After processing (involving 793 popular GitHub projects), 98,283 commit files (i.e. diff files) are obtained overall. We discover that there were 342,148 modified java files involved in these 98,283 commits. This is because a commit may have several modified files. The time distribution of these commits are shown in Fig. 6. In the following, we call the modified code snippets after the commit involved in the java classes corresponding to the project commits are referred to "CS-GC" (i.e. the abbreviation of modified code snippets involved in the commits of GitHub projects). In addition, we call the code snippets of the SO posts "CS-SO".

3. Methodology

The data we collect from SO and GitHub have been introduced in the above data collection step. Next, we adopt an efficient clone detection algorithm CCFinder (Kamiya et al., 2002) and time constraints to determine the code reuse relationship between SO and GitHub projects. The overview of our method for detecting code reuse is shown in Fig. 7.

The occurrence of identical or similar code snippets between lines of code is referred to as code cloning. Code cloning activity happens frequently. For instance, some developers use the source code of other open-source projects and incorporate it into their projects to increase development efficiency. Similar to their research (Abdalkareem et al., 2017; An et al., 2017), we identify a potential code reuse relationship of the code snippets between Stack Overflow and the commits of GitHub based on two conditions: these two pieces of code ought to be recognized as a code clone pair, and the second one's time ought to be later than the first one's. So we refer to the code pair that satisfies the aforementioned two requirements as a code reuse pair. The following four categories can be used to categorize clone types (Roy et al., 2009):

(1) Type-I: Only spaces, comments, and typography differ between the two bits of code;
(2) Type-II: Except for some identifier names, the two code snippets are identical;
(3) Type-III: The two code snippets differ in that some sentences have been added, removed, or rearranged;
(4) Type-IV: Despite having various structural variations, the two code snippets perform similar tasks.

According to certain findings from earlier studies, a sizeable component of the software system's code (usually 9% to 17%) is composed of cloned code (Zibran et al., 2011) and the percentage of cloned code in the dataset varies between 5% to 50% (Rieger et al., 2004). One statistical survey (Lopes et al., 2017) of the GitHub open-source community reveals that 70% of the code there is made up of copies of previously produced files. Their analysis revealed that at least 80% of the code files in 9% to 31% of projects might be located elsewhere.

The code on SO typically takes the form of snippets. For instance, while providing an answer, the respondent does not typically include their own code's intricate variable declarations and inheritance ties. Because of this, it is challenging to identify using cloning detection algorithms based on syntax. With the help of the analysis results of the research work of Lotter et al. (2018), they compared and analyzed several popular clone detection tools, including NiCad (Cordy and Roy, 2011), SourcererCC (Sajnani et al., 2016) and CCFinder (Kamiya et al., 2002), and finally they chose CCFinder given its performance and popularity among researchers. CCFinder is a token-based technique for clone detection and is computationally more efficient than alternative methods and has a high recall rate, and can detect all hidden clones (Roy et al., 2009). Therefore we leverage the clone detection tool CCFinder (Kamiya et al., 2002) in our study, which can detect type I and type II clone types based on token-level detection. The code snippet we detect is not complete (i.e., we need to detect the code clone between the incomplete code snippets from SO and commits). Most of Type III and IV detection tools support complete and compilable code. CCFinder is a token-based code clone detection tool which can detect incomplete code snippets. Therefore, it is difficult to detect by cloning detection algorithms based on syntax, such as some complex methods based on abstract syntax trees (Baxter et al., 1998; Tairas and Gray, 2006) or program dependency graphs (Komondoor and Horwitz, 2001; Zou et al., 2020). Moreover, our data volume is relatively large, and with some learning-based methods (Guo et al., 2020; Yu et al., 2019), the processing speed will be very inefficient. First, each line's initial indentation is removed by CCFinder, then spaces are added in between names and punctuation. Once all the identifiers have been replaced with a single token character, such as "$p" the sequence is then compared.

From the definitions of these four code cloning types, Type-I/II clones are the two most common code cloning methods, and they prefer direct copying. Wu et al. (2019) conducted an exploratory study on 321 Stack Overflow links in 289 source code files of 182 open-source

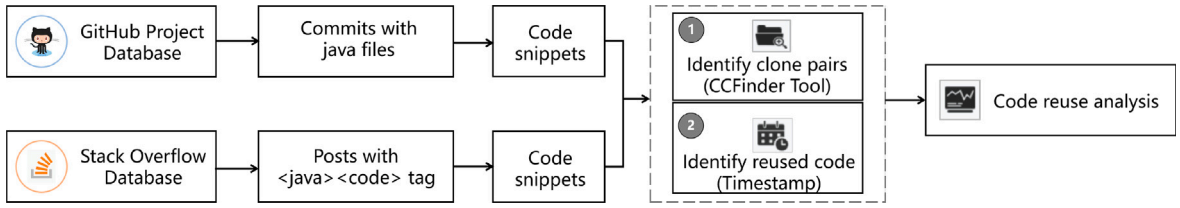

Fig. 7. An overview of our approach of code reuse detection.

```java
public int[] twoSum(int[] nums, int target) {
    int n = nums.length;
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (nums[i] + nums[j] == target) {
                return new int[]{i, j};
            }
        }
    }
    return new int[0];
}
```

(a)

```java
public int[] twoSum(int[] nums, int target) {
    int n=nums.length;
    // for loop
    for (int i=0; i<n; ++i) {
        for (int j=i+1; j<n; ++j) {
            if (nums[i]+nums[j]==target) return new int[]{i, j};

        }
    }
    return new int[0];
}
```

(b)

Fig. 8. Type-I code clone pair.

```java
public boolean isPalindrome(int x) {
    if (x < 0 || (x % 10 == 0 && x != 0)) {
        return false;
    }
    int revertedNumber = 0;
    while (x > revertedNumber) {
        revertedNumber = revertedNumber * 10 + x % 10;
        x /= 10;
    }
    return x == revertedNumber || x == revertedNumber / 10;
}
```

(a)

```java
public boolean isPalindrome(int num) {
    if (num < 0 || (num % 10 == 0 && num != 0)) {
        return false;
    }
    int rnum = 0;
    while (num > rnum) {
        rnum = rnum * 10 + num % 10;
        num /= 10;
    }
    return num == rnum || num == rnum / 10;
}
```

(b)

Fig. 9. Type-II code clone pair.

projects and demonstrated that 52% of code reused by programmers on SO is either directly copied, pasted or merely modified, which demonstrates the applicability of the CCFinder clone detection technology we use. It may not be their original intention for programmers to make heavy changes to the copied code (i.e., Type-III/IV), because there may be a tradeoff between making heavy changes to the reused code and completely rewriting a new code. Therefore, we only focus on TypeI/II clones. Figs. 8 and 9 show the clone types that CCFinder can detect, namely type-I and type-II clones. Due to the fact that the code snippets in Fig. 8 only differ slightly in how they treat spaces and blank lines, CCFinder can successfully detect them. Even though several of the variable names in Fig. 9 have been altered by programmers, CCFinder can still recognize them following preprocessing conversion.

The code snippets in the commits and the code snippets of SO will be input into the CCFinder during the specialized code reuse detection process in order to discover code clones. Since the detection method may be hard to find code clone accurately if the code snippet is very short. In the process of extracting the modified code snippets of the commits using the Change Distilling algorithm (Fluri et al., 2007), we filter out the code snippets with less than 10 tokens after extraction because longer lines of code are better at containing the current contextual information of the code, their clone detection will be more precise than that of short code snippets as input. After counting all of the extracted code snippets, we discover that the average number of tokens in each code snippet is 25. The code snippets in SO are filtered according to this standard. The CCFinder's parameters are configured using the default values.

After finishing the clone detection by CCFinder, we can get all the clone pairs. We need to filter out some clone pairs that do not meet the time limit. We require that the release time of the commit for each clone pair is later than the time of the post on SO, allowing the modified code snippets in the commits to reuse the code from SO. Reusing SO knowledge from the future in an actual development context is obviously illogical.

In addition, we discover that many cloned code snippets exist in the form of "one-to-many". A certain code snippet involved in the commits may have a clone relationship with several SO code snippets. This is due to the fact that the same snippet appears in the answers to multiple questions, or the answers to the same question are sometimes relatively similar. We choose the longest matching as the mapping result in light of this circumstance. In the absence of a similarity percentage, it is commonly accepted that the similarity is larger when there is more token segment overlap between two pieces of code (Wang and Dong, 2020).

4. Experiment

In this section, we answer our five research questions:

RQ1. How active is code reuse in development? (Section 4.1)

RQ2. What is the relationship between programmer experience and reuse ratio? (Section 4.2)

RQ3. Are code snippets reused in bug-related changes more likely to have been reused from SO? (Section 4.3)

RQ4. Are reused code snippets present in commits more likely to undergo repeated modifications? (Section 4.4)

RQ5. How do programmers behave when they reuse code? (Section 4.5)

In each research question, we will present the motivation, the approach, the results, and the practical insights.

4.1. How active is code reuse in development? (RQ1)

4.1.1. Motivation

In recent years, with the rise of the Q&A website (i.e. SO), more and more programmers integrate relevant code snippets from the SO website into their projects in order to quickly develop software projects. The purpose of our research is to explore the code reuse activity of programmers on SO in the process of software project evolution. In RQ1, we explore how the code reuse ratio changes over the years at the project-level granularity, and whether projects that are recently started will have more code reuse than those created later.

4.1.2. Approach

Determining how to identify code reuse pairs between SO and GitHub has been introduced in Section 3. Specifically, we use the year of the commit of each project as the unit to count the distribution of the code reuse ratio of the projects in each year, and we count the number of commits in each year as a comparison.

Based on the clone identification results from the earlier stage, for the calculation of the code reuse rate, we first calculate the code reuse ratio of each project in each year and then average the code reuse ratio of all projects in each year as the code reuse ratio of the projects in that year. The code reuse ratio calculation formula for each year is as formula (1), where R_i is the number of reused CS-GC in one project in a year, All_i is the number of CS-GC in one project in a year.

$$Code\ reuse\ ratio = \frac{\sum_{i=1}^{n} \frac{R_i}{All_i}}{n} \tag{1}$$

We perform the Cox Stuart trend test (Cox and Stuart, 1955) for the average code reuse ratio. The Cox-Stuart trend test is a non-parametric statistical method used to test whether there is a trend in ordered observations. This test applies primarily to time series data, but can also be used to test trends in other ordinal data. Specifically, the Cox-Stuart trend test calculates the number of positive trends and negative trends, and then uses statistical testing methods to determine whether there is a significant difference between the two. The basic steps of the Cox-Stuart trend test: (1) Set hypotheses: null hypothesis H0 (there is no trend in the data), alternative hypothesis H1 (there is a trend in the data); (2) Arrange the observed values: arrange the observed values according to time arrange in order. (3) Calculate the number of positive trends and negative trends: a positive trend refers to the situation where the latter observation value is greater than the previous observation value, and a negative trend refers to the situation where the latter observation value is smaller than the previous observation value. (4) Calculate the statistic: the statistic is usually the absolute value of the difference between the positive trend and the negative trend. This can be expressed by the formula: $C = |S_+ - S_-|$. (S_+ is a statistic that measures the positive trend in the data, and S_- is a statistic that measures the negative trend in the data). (5) Perform a significance test: according to the selected significance level, the statistics is compared with the corresponding critical value. If the statistic is greater than the critical value, you can reject the null hypothesis and conclude that there is a trend in the data. (6) Determine the direction of the trend: when the number of positive trends is greater than the number of negative trends, it means there is a positive trend; otherwise, there is a negative trend.

The reasons why we utilized the Cox-Stuart trend test can be attributed to the following points: (1) Non-parametric statistical methods. (2) Suitable for ordered data. Since trends usually involve ordered time series data, the Cox-Stuart trend test is designed to handle ordered observations. This makes it particularly suitable for time-related data, such as we use the Cox-Stuart trend test for changes in code reuse rate over time. (3) Explainable. The Cox-Stuart trend test is relatively simpler than some complex trend analysis methods. It provides an intuitive way to determine whether a trend exists in the data while avoiding some of the assumptions and parameters required by complex models. (4) Directional information. Not only can you test whether a trend exists in your data, but you can also determine the direction of the trend (positive or negative). This is valuable in understanding whether code reuse is gradually increasing or decreasing. (5) Suitable for small samples. For smaller data sets, the Cox-Stuart trend test may be more suitable than some methods that require large amounts of data support.

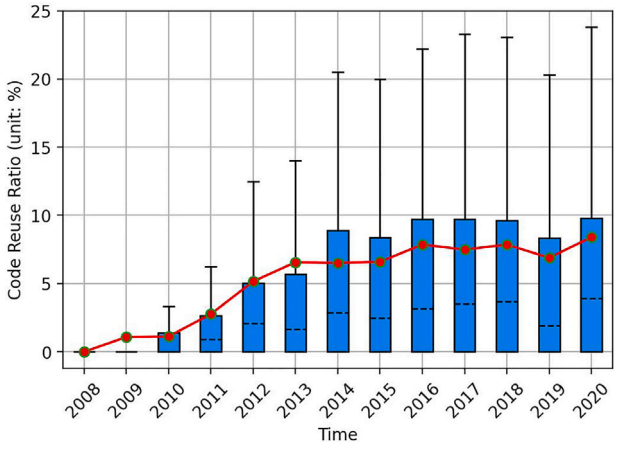

Fig. 10. The code reuse ratio with its trend over the years. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

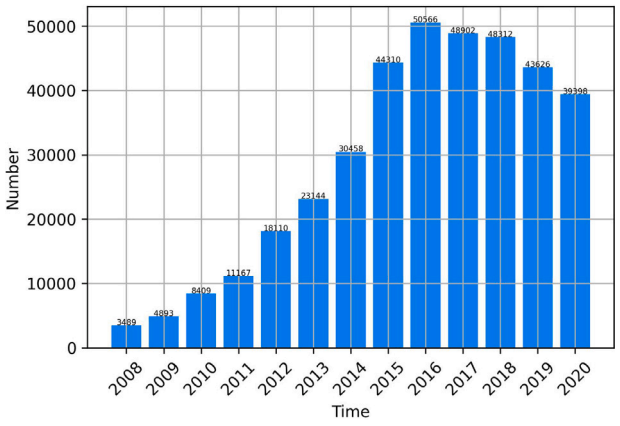

Fig. 11. Distribution of the number of projects with commits by year.

4.1.3. Results

The average range of code reuse rate distribution each year in the open-source code database ranges from 0% to 8.4%. The average code reuse ratio across all projects is 6.32%, demonstrating that developers may make use of the information on the SO Q&A website during the project's iterative development process. The red point in Fig. 10 presents the mean, and the dotted line represents the median (the code reuse ratio is larger than its maximum value in Fig. 10 because some outlier points are too high and affect the display of the figure, they are not shown in the figure).

From Fig. 11, the number of projects committed after 2016 is on a downward trend. Fig. 10 shows that the code reuse ratio increase year over year from 2008 to 2016 and climbed consistently after 2016. Perhaps because we collect a fixed number of projects (i.e. 793 projects), some projects have submissions at a later stage but the projects gradually mature and stabilize. Even if the latter trend has leveled off, the overall trend is still rising. We perform the Cox Stuart trend test (Cox and Stuart, 1955) for the average code reuse ratio. The result shows that p-value is 0.03 which is lower than 0.05 and S_+ is

higher than S_- (S_+ is a statistic that measures the positive trend in the data, and S_- is a statistic that measures the negative trend in the data), which shows that the code reuse ratio will have a significant upward trend in the future. This demonstrates once again how developers tend to reuse the information found on the SO Q&A website, and how this tendency is growing more and more common over time as the SO Q&A website has gained popularity.

In general, from the above analysis results, our study shows that the code reuse ratio is likely to grow in the future. A code snippet from SO will inevitably be utilized several times in the same or other projects. Then, if the code snippet from SO is defective, the defect may spread to several projects. Therefore, our practical insight is that locating the symmetrical vulnerabilities (Zhang et al., 2015) in the same or different projects is a worthy research direction for researchers.

4.2. What is the relationship between programmer experience and reuse ratio? (RQ2)

4.2.1. Motivation

When working on a project, some experienced developers might be better at finding the programming knowledge they need on SO because they have a firm grasp of some professional knowledge, whereas some beginners might not have enough familiarity with some programming knowledge or professional terminology, which could limit their ability to find programming knowledge. This research question is being posed with this motivation in mind.

4.2.2. Approach

The level of experience of programmers is based on programmers who submit GitHub. It is not enough to only rely on programming years to measure the level of experience, because some programmers do not carry out any development or submission process after creating a GitHub account. Moreover, we cannot obtain data on the programming year of programmers. Therefore, we use the number of submissions (i.e. the commit count) made by programmers in GitHub to measure the programming experience of developers.

Based on how many commits a committer (i.e., developers) has made to the project, we gauge their level of experience. We presume that a developer's project experience increases with the number of contributions they make. We calculate the proportion of all committers with different programming experience levels. Additionally, we track the percentage of committers with various levels of experience who reuse code. Further, we separately explore the post attention of posts reused by programmers with different levels of experience. Specifically, we measure the post attention according to the metrics in the post information: FavoriteCount, Score, CommentCount, AnswerCount.

4.2.3. Results

Since most of the project contributors submit less than 100 commits and in order to make the number of people in each category as balanced as possible, we divide them into six categories in Table 2 based on their levels of project experience. Table 2 makes it quite evident that a bigger percentage of committers who have reused knowledge on SO are those who have more experience. This may mean that if the developers are familiar with the project, they will be better able to use their understanding of SO to address issues that arise. Therefore, our practical insight is that the SO platform should promote its website to inexperienced developers.

Further, we count the number of posts reused by programmers with different levels of experience. As can be seen from Fig. 12, the CommentCount indicator and the AnswerCount indicator do not change much with the level of programmer experience. We use the Pearson correlation coefficient calculation method to calculate the correlation coefficient value of the indicator and interval (the range is [−1,1]). The correlation coefficients of the CommentCount indicator and the AnswerCount indicator with the programmer's experience level range

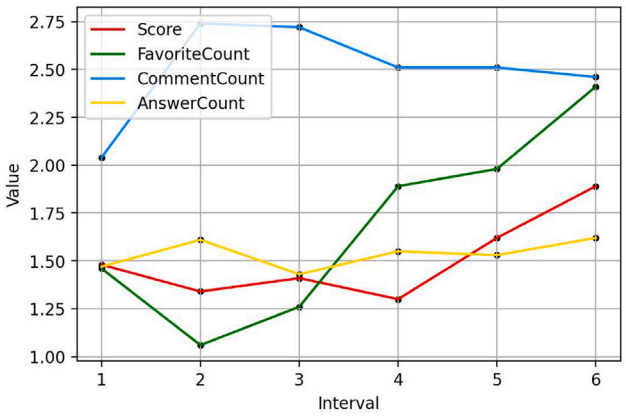

Fig. 12. The relationship between the post attention and the experience of the programmer.

Table 2
Code reuse ratio of programmers with different levels of experience (from the perspective of the committer).

Interval	Commit count	Committer count	Committer count (reuse SO)	Ratio
1	1	2630	165	6.27%
2	2	1275	140	10.98%
3	3–5	1776	341	19.20%
4	6–15	1773	694	39.14%
5	16–100	1659	1205	72.63%
6	>100	657	633	96.34%

(i.e. [1,2,3,4,5,6]) are 0.3 and 0.4 respectively. Therefore, we believe that the comment Count indicator and the AnswerCount indicator do not change much with the level of programmer experience.

However, we use the Pearson correlation coefficient calculation method to calculate the correlation coefficient value of the indicator and interval (the range is [−1,1]). The correlation coefficients of the Score indicator and the FavoriteCount indicator with the programmer's experience level range (i.e. [1,2,3,4,5,6]) are 0.7 and 0.9 respectively. Therefore, we believe that the Score indicator and the FavoriteCount indicator of the post increase as the programmer's experience level increases.

It shows that programmers with high experience are more inclined to reuse posts with higher scores and a larger number of collections which means the FavoriteCount indicator is high. This can also provide some suggestions for novice programmers: when reusing knowledge on SO, try to reuse some high-attention posts to better learn the knowledge to address the issues they encounter.

4.3. Are code snippets reused in bug-related changes more likely to have been reused from SO? (RQ3)

4.3.1. Motivation

Some developers may visit some Q&A websites, like the SO platform, to look for assistance or useful knowledge when they run into bug-related problems while working on development projects. In order to analyze if code snippets used in bug-related changes are more likely to be reused from SO, we conduct RQ3. We want to further explore whether the posts reused by these bug-related commits are different from those reused by non-bug-related commits, and what is the distribution of the main types of these posts. We want to analyze whether it is possible to reuse the same post in commits that fix different bugs.

4.3.2. Approach

Each CS-GC has a corresponding commit message in the open-source project code database. We use the terminology associated with bugs to

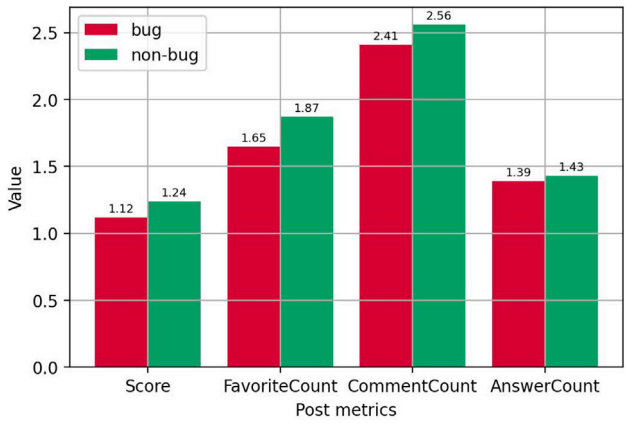

Fig. 13. The average value of the metrics for the posts that are reused by bug-related and non-bug-related commits.

determine whether a commit is connected to the bug (i.e. Some words related to bugs: bug, fix, solve, issue, problem, error, repair, defect, vulnerable, vulnerability). In order to verify the quality of this method, we randomly select 100 buggy commits for manual judgment, and 93 of them can be confirmed to be real buggy. After separating the commits into bug-related and non-bug-related commits, we count the number of bug-related CS-GCs and non-bug-related CS-GCs that reuse the code snippets on SO based on the results of the CCFinder algorithm's clone detection and the chronological order of the SO post and the commit.

To investigate whether the post attention of posts reused by bug-related commits is different from that of posts reused by non-bug-related commits, we adopt the same method as in RQ2, according to the indicators (i.e. FavoriteCount, Score, CommentCount, AnswerCount) in the post information to measure the post attention. In addition, we mine the semantic information of posts based on the textual information of their titles and tags to compare the differences of the reused posts (by the same commit) in the semantic space of the text. Since Word2vec (Mikolov et al., 2013) which is widely used in various semantic analysis tasks, such as text classification tasks (Zhang et al., 2018; Liu et al., 2017), semantic analysis (Lin et al., 2015; Shen et al., 2018) is a lightweight semantic extraction model and can extract contextual semantic information well. We also consider the more popular transformer-based models (e.g. BERT Devlin et al., 2019) to extract the semantic features, but transformer-based model is relatively large and inefficient. Moreover, when the questioner asks a question on Stack Overflow, they need to compose the title and tags of the post, where the title cannot be longer than 150 characters and the number of tags can be no more than 5. When the questioner describes the title of the question, he may use some relatively concise sentences to describe the problem, and because tags are composed of different words (they do not have strong contextual semantic information like sentences). Therefore, it is more appropriate to use Word2vec for semantic representation at this time than BERT, Word2vec is more focused on capturing word-level semantic information because it is trained in word units. Using Word2vec is a suitable choice for our scenario to semantically represent titles and tags the tasks that are closely related to word-level relationships. Therefore, we leverage Word2vec model to extract the semantics features. The two methods used to train Word2vec models are Skip-gram and CBOW. We adopt the commonly used CBOW training method.

We average the embedding vectors of all words as the embedding vector of the sentence. These word vectors are often high-dimensional dense vectors (for example, we use 300 dimensions here). Specifically, we utilize the titles of all posts to construct a corpus for self-supervised training. In order to visualize the distribution of these semantic vectors in low dimensions, we adopt the PCA (principal components analysis) method proposed by Hotelling (1933), which is a linear transformation method. It is usually used for dimensionality reduction of data. While reducing dimensionality, the feature with the largest contribution to the variance of the data is maintained, that is, the least dimension is selected to summarize the most important features. Each new variable is a linear combination of the original variables and the new variable is linearly independent.

Table 3
Statistics of code reuse ratio of bug-related and non-bug-related CS-GC.

	Bug-related CS-GC	Non-bug-related CS-GC	All CS-GC
CS-GC count	57,252	284,896	342,148
CS-GC (reuse SO) count	3636	17,986	21,622
Code reuse ratio	6.35%	6.31%	6.32%

To explore the distribution of types of posts reused by bug-related commits, we employ the Latent Dirichlet Allocation (LDA) topic model to cluster and extract topics for each category.

To use LDA for topic analysis, the number of topics needs to be pre-specified. We use an indicator commonly used in LDA model analysis to determine the optimal number of topics: topic coherence (Röder et al., 2015). The higher the indicator, the better.

In addition, we explore in detail the posts that are frequently reused by bug-related commits, and the commits corresponding to these posts.

4.3.3. Results

According to the findings of Table 3, the code reuse ratio of the bug-related CS-GC is slightly higher than that of the non-bug-related CS-GC, which may indicate that developers should not directly reuse code from SO but assess its security risk. This also demonstrates that developers do not just apply what they have reused from SO to fix bugs. Developers may reuse the information from SO, for instance, when they add new features or optimize the code structure which is not related to bug fixing.

From Fig. 13, in terms of these four indicators (i.e. the Score indicator, the FavoriteCount indicator, the CommentCount indicator, and the AnswerCount indicator) of the post, the posts reused by bug-related commits are lower than the posts reused by non-bug-related commits.

To further utilize the semantic information of the post, we adopt Word2vec to convert the textual information of the post into dense high-dimensional vectors and use the PCA method to reduce these high-dimensional vectors to two-dimensional visualization as shown in Fig. 14. The distribution of Word2vec vectors of posts reused by bug-related commits (3636) and posts reused by non-bug-related commits (17,986) are different, which shows that the posts reused by these two types are different.

Then we use LDA to cluster these posts (that is, topic analysis). First, we need to determine the optimal number of topics. We use the coherence indicator (Röder et al., 2015) as our reference for choosing the appropriate number of topics. The number of reference topics calculated by this indicator is 20. However, it can be seen from Fig. 15 that among the 20 classes, there are still many classes that are relatively compact and have overlapping parts. We decrease the number of topics from 20 to 1 until there is no intersection between the topics and choose an appropriate coherence indicator. Finally, we determine that the optimal number of topics is 4.

From Figs. 16–19, we can see the distribution of the types of 4 topics, which can be summarized into the 4 topics shown in Table 4. It can also be seen that the types of posts reused by bug-related commits are mainly distributed in these four topics: related to web development (topic I), related to GUI (swing), android (topic II), related to application development (spring, spring-boot framework) (topic III), related to database (hibernate, MySQL, MongoDB) (topic IV). We recommend

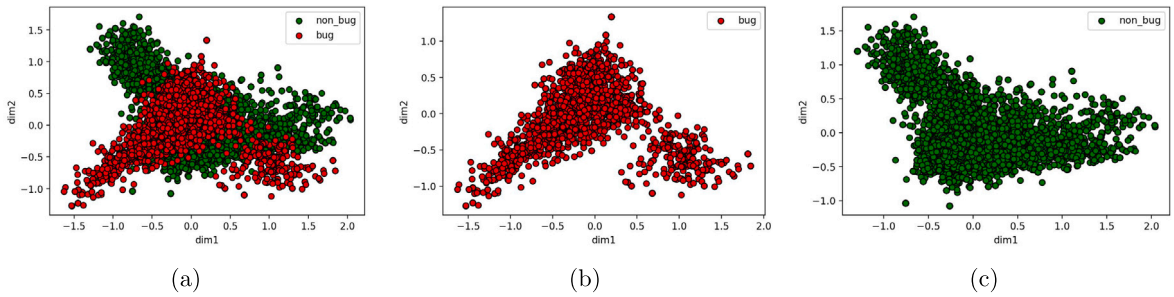

Fig. 14. Semantic feature distribution of posts after PCA dimensionality reduction.

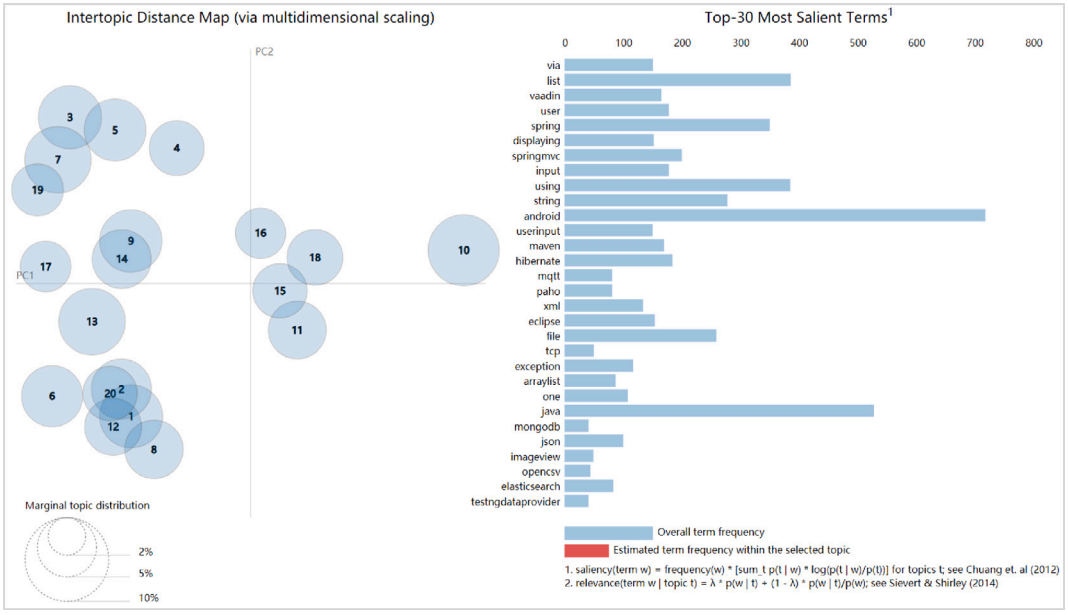

Fig. 15. LDA topic clustering (the number of topics is 20).

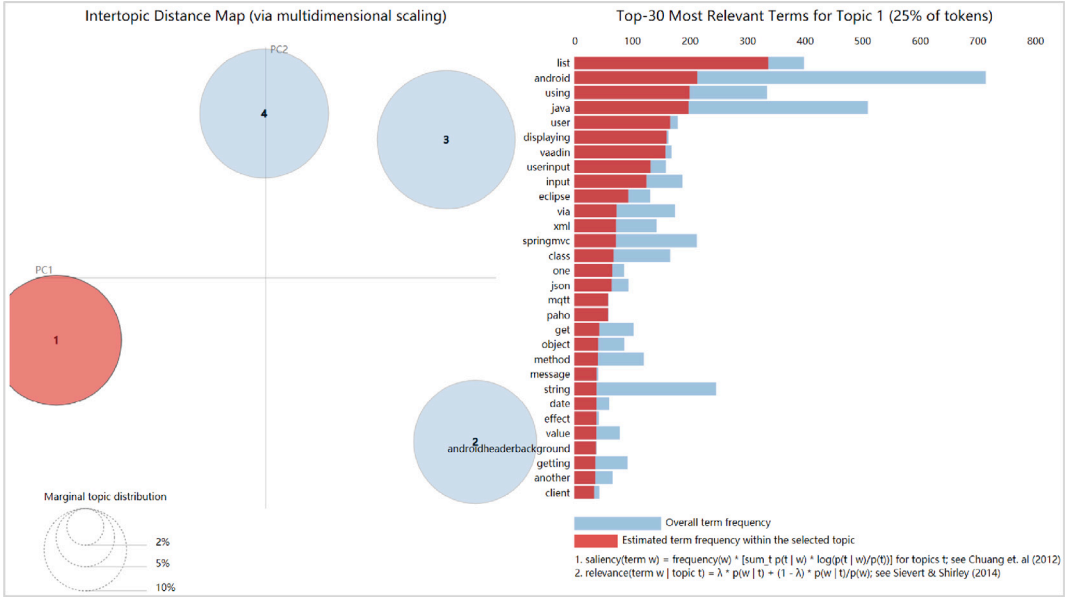

Fig. 16. Topic I.

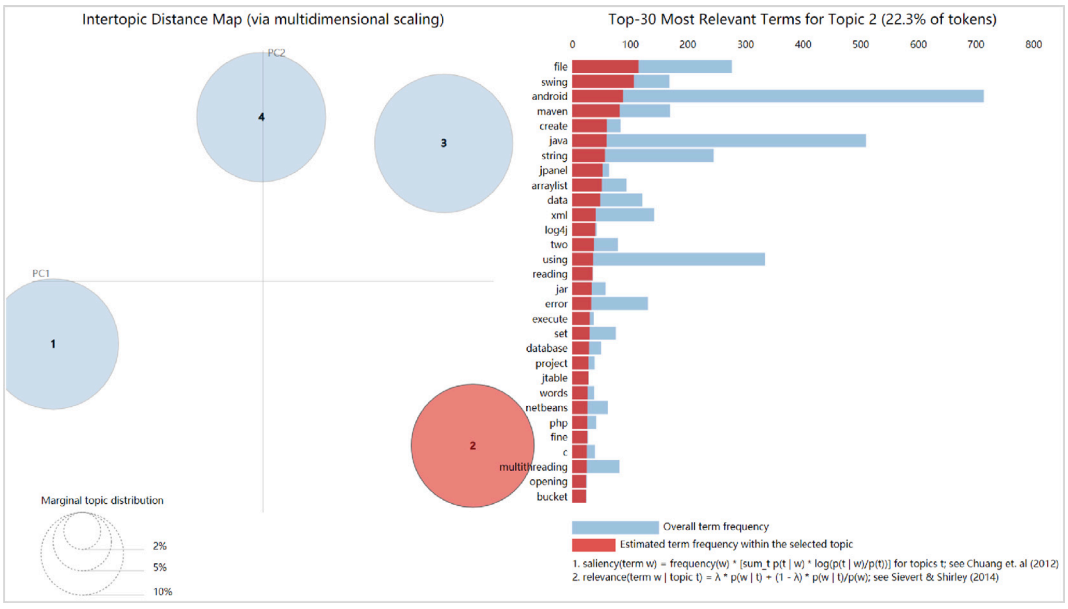

Fig. 17. Topic II.

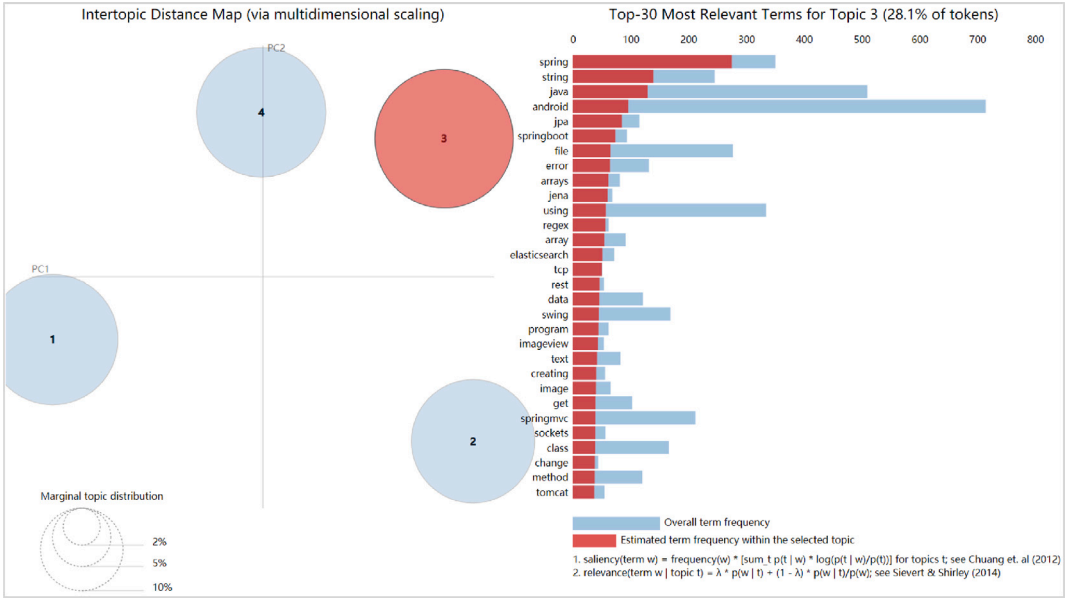

Fig. 18. Topic III.

Table 4
Summary of LDA Topics.

Topic	Description
Topic I	Related to list data structure, web development (vaadin)
Topic II	Related to file, GUI (swing), android
Topic III	Related to application development (spring, spring-boot framework)
Topic IV	Related to database (hibernate, MySQL, MongoDB)

Table 5
Statistics of the number of times the post be reused.

Interval	Posts count
1	1291
[2, 20)	541
[20, 40)	7
[40, 60)	1
[60, 80)	1
[80, 116]	1

that programmers pay attention to the post attention and safety of the code when conducting these four types of development research.

Further, we select those posts that are reused multiple times by bug-related commits. There are a total of 1842 posts corresponding to bug-related commits. Table 5 shows the statistics results according to the number of these posts reused by bug-related commits in 6 intervals (because the number of posts that are reused once and multiple times is quite different, and in order to highlight some extreme values, we divide the interval in this form).

We select the posts reused by 65 bug-related commits, and their corresponding bug-related commits. The post information is shown in Fig. 20, and the commit message of these bug-related commits is shown in Fig. 22, and it can be seen from the cloud map (Fig. 21) that the

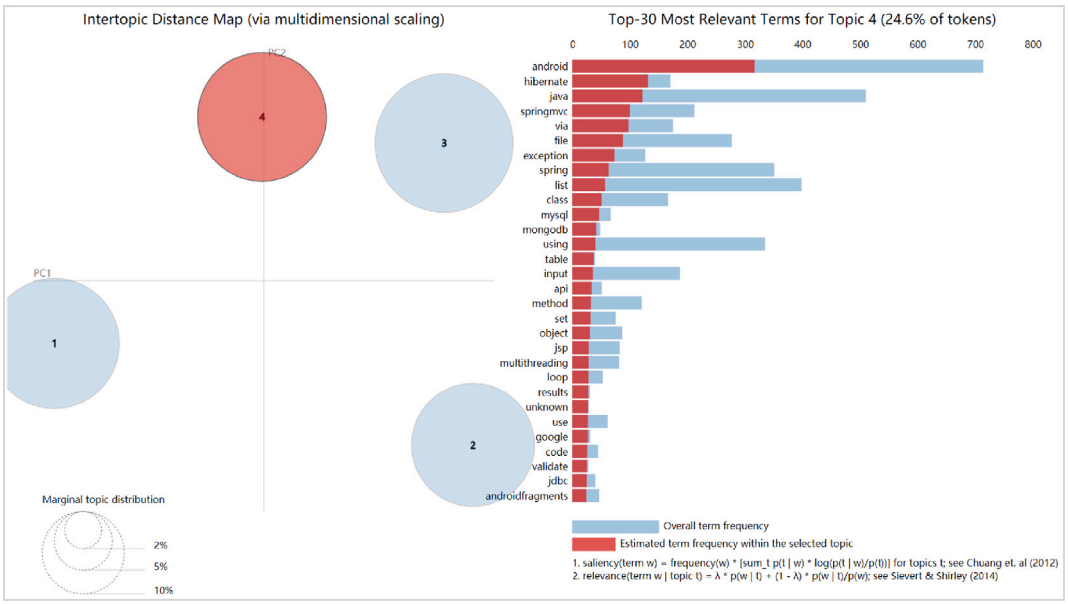

Fig. 19. Topic IV.

Reused by 65 bug-related commits:

CreationDate = 2016-11-04T22:02:58.530
Tags = <java><android><mqtt><mosquitto><paho>
Title = How to publish only one message using a Java MQTT client using Eclipse Paho
Score = 1, **AnswerCount** = 2, **CommentCount** = 0

Fig. 20. The post reused by 65 bug-related commits.

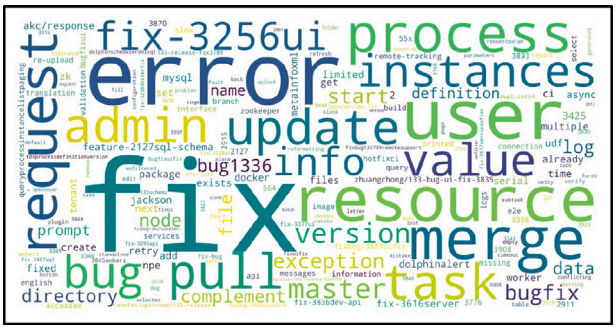

Fig. 21. The world cloud map of the bug-related commits.

bugs fixed by these bug-related commits are diverse, some of them are fixing the same type of bugs and some are different, which shows that these different bug-related commits reuse the same post. In addition, the above results also indicate that when programmers fix different bugs, they may abstract the problems they encounter into the natural language to search, which may lead to the above result, that is, to fix different bugs but reuse the same post.

4.4. Are reused code snippets present in commits more likely to undergo repeated modifications? (RQ4)

4.4.1. Motivation

Developers may reuse code from Q&A websites for developing software projects. These reused codes may require custom changes to adapt to the native program context, and some code snippets are very long and the security of the code on the Q&A website is unknown, which may cause developers to make multiple modifications to these codes. In light of this purpose, we seek to determine whether the reused code snippets present in the commits are more likely to undergo repeated modifications. (i.e. from this perspective, we investigate whether the reused code snippets are more likely to have security risks). Further, we study whether low-attention posts might cause programmers to make multiple commits in the process of reusing code, and reuse the same posts repeatedly.

4.4.2. Approach

First, we extract the code snippets involved in the java classes that have been modified multiple times corresponding to the project commits. Then, we perform clone detection on these code snippets with the code snippets of the SO posts. We use the CCFinder clone detection algorithm to perform on CS-GC, and then determine whether the CS-GC belongs to the same class in the same project. If they do, the assumption is that the java class has undergone many modifications. Additionally, we separate the class files that have undergone more than one modification into two groups: those with reused SO posts and those without, and then separately count the code reuse ratios corresponding to the class files under consideration. The CCFinder algorithm is used in the following phase to input CS-GC and CS-SO for code clone detection. In order to accurately determine whether low-attention posts may cause programmers to commit multiple revisions in the process of reusing code, and reuse the same post repeatedly, we need to confirm the code snippets that have been modified multiple times cannot be similar. If they are similar, the reused posts may also be the same, so we use CCFinder to perform clone detection on the code snippets involved in these commits to ensure that the class files that have been modified multiple times are modified differently, and reuse the same post.

4.4.3. Results

The experimental results are shown in Table 6, the code reuse ratio (14.44%) in the code snippets involved in the java classes that have been modified multiple times corresponding to the project commits (i.e., the number of modifications is more than once) is significantly higher than that in the overall code reuse ratio (6.32%). The code reuse ratio has essentially remained at 12% to 15% despite the larger number of modifications, which indicates that the java files involved in

Commit messages of the bug-related commits

- 'FIX-3836][dev-API] process definition validation name interface prompt information error (#3908)',
- '[bugFix][ui] Cannot select connection',
- '[Fix-3256][ui] Fix admin user info update error (#3306)',
- '[Feature-2127][sql-schema] Fix the mysql version limited error (#2127) (#2915)',
- ' [Fix-3616][Server] when worker akc/response master exception , async retry (#3776)',
- 'Merge pull request #3870 from zhuangchong/1.3.3-bug-ui-fix-3835',
- 'fix Zookeeper does not start, printed logs have no error messages, Api services can start but cannot be accessed',
- '[Feature-2127][sql-schema] Fix the mysql version limited error for 5.5x - 5.64 (#2911)',
- 'fixed queryProcessInstanceListPaging query slow (#3893)',
- '[Hotfix][ci] Fix e2e ci docker image build error',
- '[fixbug-3887][ui]Fix missing English translation of re-upload files',
- "Merge remote-tracking branch 'remotes/upstream/1.3.3-release' into 1.3.3-release-fix#3789",
- '[Fix-3077][ui] Fix the edit name duplicated verify (#3346)',
- "[FIX-Bug #3845][Ambari Plugin] Start Ambari report an error: Table 't_ds_process_definition_version' already exists (#3846)",
- '[Fix-3256][ui] Fix admin user info update error (#3425)',
- '[FIX_BUG_1.3#3789][remote]support netty heart beat (#3868)',
- 'fix zk path error (#3470)',
- '[bugfix] remove Conflicting configuration sudo and user (#3038)',
- '[Fix-3616][Server] when worker akc/response master exception , async retry (#3748)',
- 'Fix unfiltered jar files in resource directory and UDF upload resource parameters (#3008)',
- '[Fix-#3487][sql] add dolphinscheduler_dml.sql under 1.3.3_schema (#3907)',
- 'Merge pull request #3923 from lgcareer/dev-fix',
- '[Fix-3238][docker]Fix that can not create folder in docker with standalone mode',
- . . .

Fig. 22. The commit message of the bug-related commits.

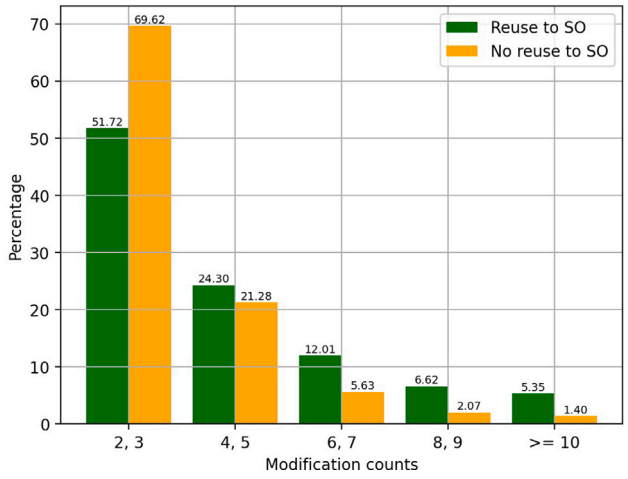

Fig. 23. The distribution of java classes involved in the commits that reuse CS-SO and not reuse CS-SO.

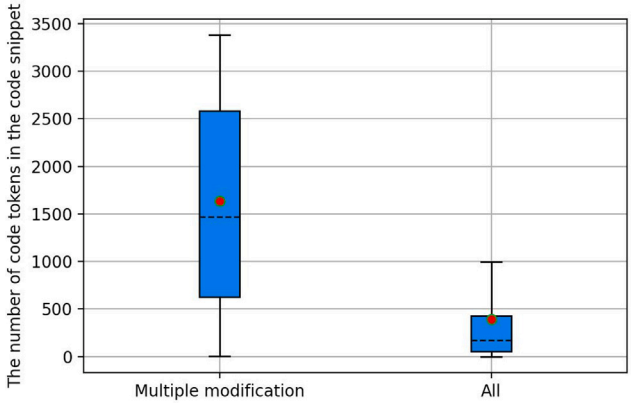

Fig. 24. The number of code tokens of the code snippets of the posts reused by the java classes involved in the commits that have been modified multiple times and the total posts.

Table 6
Code reuse ratio corresponding to different modification counts.

Modification count	Commit count	Commit (reuse SO) count	Code reuse ratio
1	309,778	16,948	5.47%
2, 3	13,536	2137	15.78%
4, 5	9026	1168	12.94%
6, 7	4322	622	14.39%
8, 9	2283	322	13.26%
≥10	3203	425	14.10%

commits with code reuse are more likely to have undergone repeated modifications by developers.

The comparison results in Fig. 23 demonstrate that classes with reusing SO posts have a larger proportion than classes without reusing SO posts (except for those with modification count of 2 and 3).

Further, according to the statistics of 21,622 commits with reusing relationships, there are 6720 included java classes and 757 classes that have been modified multiple times and have been reused multiple times (greater than or equal to 2 times). Among them, 54 classes that have been modified more than once time and reuse the same post, accounting for 7.13% (54/757).

Additionally, we compare the length of the code snippets of the posts reused by the java class involved in the commits that have been modified multiple times and the total posts, as can be seen from Fig. 24, probably because the length of the code snippets in these posts are longer and the knowledge content is more complete so that the developers make multiple modifications to the same java class involved in the commits when developing and reusing these posts multiple times.

Fig. 25 shows the comparison of the post attention of the posts that have been reused by the multiple modified commits and the post attention of the total posts. It can be seen that, in terms of the average value of the four indicators of FavoriteCount, Score, CommentCount, and AnswerCount, the posts reused by the multiple modified java classes involved in the commits are lower than the total posts, which probably reveals that low-attention posts may cause programmers to make multiple revisions in the process of reusing code. And we can get the result from Fig. 24, programmers may focus more on the posts

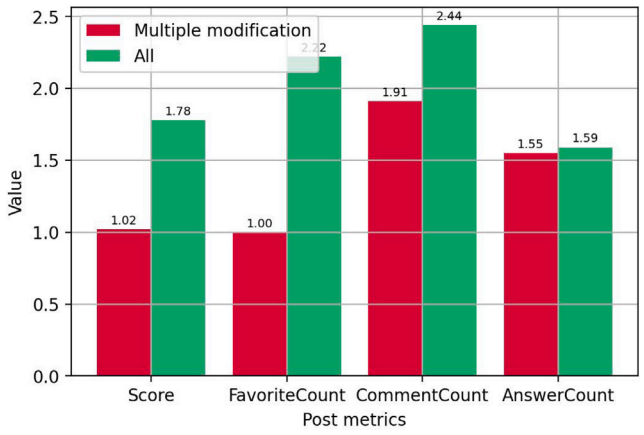

Fig. 25. The average value of the metrics for the posts that reuse by the multiple modified commits and the total posts.

where code snippets are relatively short when looking for answers. We suggest that when programmers reuse the code on SO for software project development, they should pay more attention to the post attention and quality of the posts and reuse some high-attention posts.

4.5. How do programmers behave when they reuse code? (RQ5)

4.5.1. Motivation

In this study, considering that the code snippet involved in the commit may correspond to multiple reused posts, we would like to explore whether the types of these posts are related, that is, when programmers reuse the code on SO, whether they will reuse posts of similar types.

4.5.2. Approach

For calculating the correlation of multiple reused posts corresponding to the code snippet involved in the commit, we use the sum of squared error index (SSE), which is often used in clustering algorithms. The specific calculation formula is shown in formula (2). We leverage the word2vec algorithm introduced above to embed the title and the tags of each post to obtain a vector, and the vectors in a group of posts are added and averaged as the center vector of the group of vectors, and then brought into the SSE formula to calculate the SSE value that is used as the degree of correlation between posts within each group of posts.

$$SSE = \sum_{i,k}(x_{i,k} - \mu_k). \tag{2}$$

Then we normalize the SSE value, as shown in formula (3):

$$X_{std} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{3}$$

4.5.3. Results

In order to determine the correlation of the reused posts corresponding to the code snippets involved in a commit, we calculate the normalized SSE value of each post group and divided it into five intervals as shown in Table 7, The smaller the SSE value, the more relevant the group of posts, because they are less discrete from the center point.

From Table 7, 4.77% of the post groups are quite related. We select two groups of posts in the [0, 0.2) interval for analysis, as shown in Fig. 26. We can see that in the first example, the posts are all addressing file-related issues. In the second example, the posts are all addressing hibernate-related issues. In addition, it can be seen from Table 7 that 48.79% of the post groups are distributed in the interval [0.6, 0.8), and the natural language semantic similarity between the posts in these post groups corresponding is not very high. Since we use the code snippets of posts to input Clone detection algorithms (i.e. CCFinder) for determining code reuse pairs, we further analyze whether the code snippets in these semantically unrelated posts are related. Specifically, as shown in Fig. 27, we select the code snippets of some posts in this interval(i.e. the interval [0.6, 0.8)) for comparison and find that although the natural semantics of these posts are not very related, the code snippets are very similar. The above analysis shows that when programmers reuse posts, they may abstract the content to be retrieved into texts with different meanings for retrieval, so that the retrieved posts may have different titles but similar code snippets. The above results show that programmers may refer to knowledge from multiple related posts when reusing the SO code. Therefore, we suggest that the SO platform can appropriately recommend more satisfactory answers to programmers based on the information of code snippets.

Table 7
Number of SO posts.

Interval	Count	Percentage
[0, 0.2)	777	4.77%
[0.2, 0.4)	1891	11.60%
[0.4, 0.6)	5185	31.82%
[0.6, 0.8)	7950	48.79%
[0.8, 1.0]	491	3.01%

5. Related work

5.1. Empirical studies on Stack Overflow

In recent years, the rise of the Q&A website SO has attracted the attention of a large number of researchers, and a lot of research conducted empirical studies on SO (Baltes and Treude, 2020; Baltes et al., 2017; Fischer et al., 2017; Scoccia et al., 2021; Zhang et al., 2019b; Hsieh et al., 2010; Jan et al., 2017; Chen et al., 2019; Tóth et al., 2020). Some studies focus on the issue of citation permissions and the proliferation of unsafe code on SO. Specifically, Baltes et al. (2017) find that although SO provides a large number of reproducible code snippets, most programmers do not reasonably cite SO's license when copying code knowledge on SO. Fischer et al. (2017) analyzed android applications on Google Play and code snippets on SO for the proliferation of unsafe code on SO. Their results show that 15.4% of the 1.3 million Android applications they analyzed, contained security-related code snippets from Stack Overflow. 97.9% of these have at least one unsafe piece of code in them. Zhang et al. (2019a) found the fact that it is believed that SO has amassed a wealth of software programming knowledge, some of the knowledge in the post may now be out of date. If these out-of-date responses are not processed promptly, they could mislead certain programmers who reuse them and lead to security issues. Moreover half of the outmoded responses, according to their findings, were probably already out of date when they were first posted. If programmers reuse these outdated posts, they might introduce security flaws. Meng et al. (2018) carried out an empirical analysis of SO posts and discovered that programming issues are frequently related to libraries or APIs. They also highlighted several vulnerable code suggestions. Some study focuses on dissecting the structure and operation of the SO system and offering recommendations to the SO community.

Compared with the research on SO described above, the main difference between us and them is that we focus on and discuss the analysis of code reuse on SO during the evolution of GitHub open-source project software. Our study mainly investigates these research points in terms of: (1) The code reuse ratio in open-source projects and its trend over the years, a comparison of code reuse in old and new projects. We conducted a trend analysis on code reuse ratio. (2) The relationship between developer experience and code reuse, the post

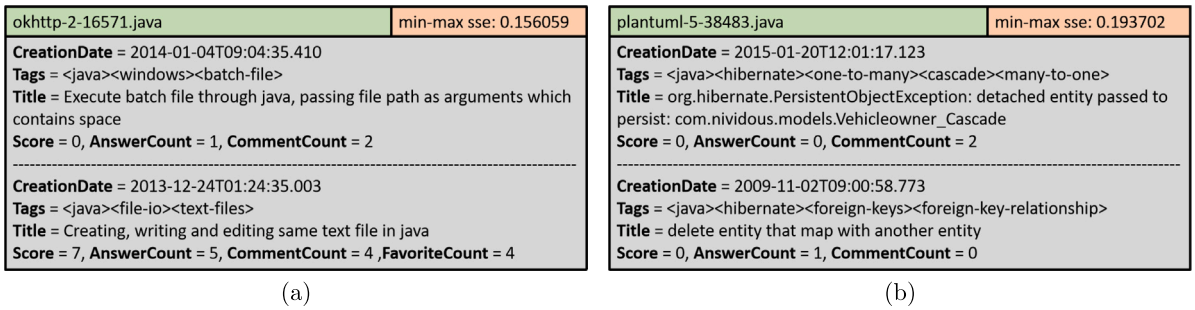

Fig. 26. Two examples with lower SSE values.

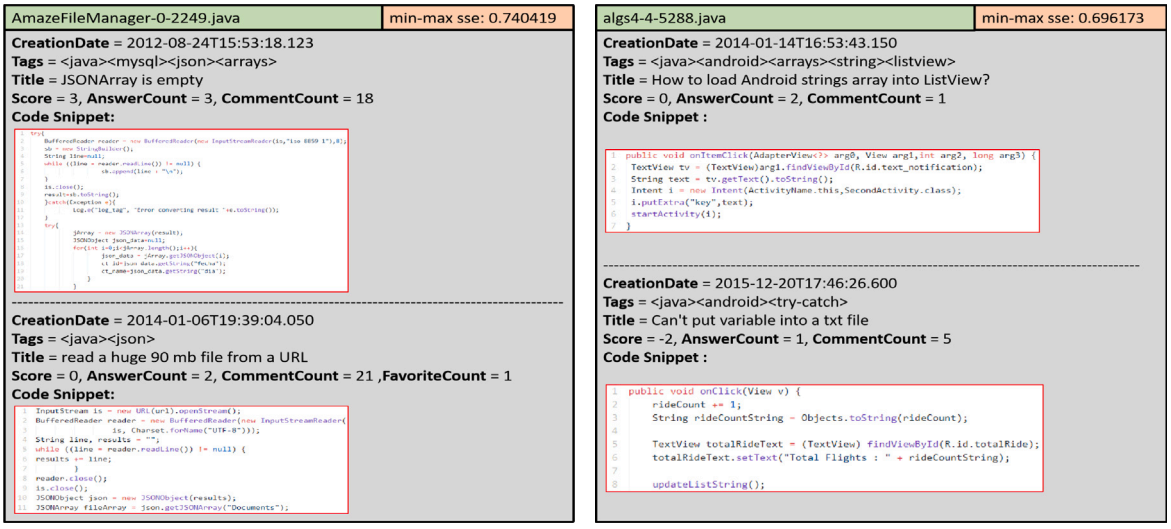

Fig. 27. Two examples with higher SSE values.

attention analysis of posts reused by programmers with different levels of experience. (3) The relationship between code reuse and modified code snippets involved in the bug-related commits, the analysis of posts reused by bug-related commits, introducing Word2vec, LDA algorithm for semantic analysis of posts. (4) Exploring whether code reuse may cause developers to modify multiple times and whether multiple modifications will reuse the same posts, and analyze these posts attention. (5) The type of SO posts that developers generally like to reuse and correlation analysis between multiple posts reused by developers.

5.2. Code reuse related to Stack Overflow

In previous studies, some researchers have also studied the code reuse activity between open-source projects and SO (Lotter et al., 2018; Yang et al., 2017; Manes and Baysal, 2019; Nishi et al., 2019; Abdalkareem et al., 2017; An et al., 2017; Ragkhitwetsagul et al., 2019; Wu et al., 2019, 2023). Lotter et al. (2018) discovered that code reuse is widespread between open-source projects and SO. They mined 151,946 java code snippets from Stack Overflow and 16,617 java files from 12 of the top weekly listed projects on SourceForge and GitHub. Their analyses are aimed at finding the number of clones (indicating reuse) (a) within Stack Overflow posts, (b) between Stack Overflow and popular java OSS projects, and (c) between the projects. Based on their result we observed that 7.6% of the files under consideration contain at least one reuse. The purpose of their research is mainly for open-source projects and SO Between complex licensing issues, to ensure the appropriateness of code reuse, and licensing requirements awareness. The research object is code reuse analysis of java files from open-source projects and code snippets on SO. They do not introduce a time constraint to determine the direction of reuse. We introduce a time constraint to determine the direction of reuse between open-source projects and SO. Wu et al. (2023) find that API tutorials and SO posts together can provide more API knowledge. Therefore, they design a deep transfer metric learning based relevance identification model to retrieve and reuse APIs from API tutorials and Stack Overflow based on natural language queries. Abdalkareem et al. (2017) carried out an exploratory investigation concentrating on mobile app code reuse from SO. In particular, they looked into who, what, when, and how much code is reused. An et al. (2017) investigated whether developers follow licensing restrictions while reusing code from SO posts in a case study involving 399 Android apps. Their findings showed that programmers might have cloned the code of applications that would have been utilized to respond to SO inquiries. They aimed to increase community awareness of potential unethical code reuse practices among software engineers. Yang et al. (2017) aimed at investigating and understanding code knowledge sharing between GitHub and SO. Their research object is the reuse between source files of python projects on GitHub and code snippets of SO posts, they adopted SourcererCC as a clone detector and high similarity code pairs as clone pairs, but they did not introduce the time factor to determine the direction of code reuse.

After comparing with the related work above, the main differences between our research and theirs can be summarized as these: (1) Our research object is different from related work. We focus on the reuse cases during the software development (i.e., commits). As a result, the related work we introduce above focuses on detecting the code reuse between the source code of a project and the code snippets in SO, while we detect the code reuse between the commits and code snippets in SO. Hence, our research objects are different, and we focus more on

code reuse in the dynamic process of software development. (2) Our research scale is different from related work. We gather 793 GitHub projects and 1,355,617 posts from SO. The 793 GitHub projects that we gather comprise different types of fields rather than concentrating on examining fewer project types. (3) Our research questions are different from related work. Furthermore, the iterative process of the project as well as the code reuse behavior of programmers are the main topics of our study. Due to the different RQs, we get different research results. Also, we get some different insights for the stakeholders.

6. Threats to validity

External Validity. Threats to external validity concern the generalizability of our findings. We focus on SO, which is one of the largest and most popular Q&A Websites. Yet, our findings may not generalize to other Q&A Websites. We study 793 open-source projects, which may not be representative of all software projects, but such a large-scale study may yield more general insights. However, our study reflects the findings from highly-used projects, making code reuse an important element to consider.

Internal Validity. The approach we employ to locate code clones is the CCfinder developed by Kamiya et al. (2002), which can only identify types I and II of clones. However, there are four types of clone detection, of which Types I, II, and III are related to syntactic cloning while Type IV is related to semantic cloning. From the definitions of these four code cloning types, Type-I/II clones are the two most common code cloning methods, and they prefer direct copying. Although our research only identifies two forms of clones, the rationale for employing the CCFinder clone detection tool is further demonstrated by the fact that Wu et al. (2019) conducted an exploratory study on 321 Stack Overflow links in 289 source code files of 182 open-source projects. They found that when programmers reuse the code on SO, 52% of it is directly copied, pasted, or simply modified. It may not be their original intention for programmers to make heavy changes to the copied code because there may be a tradeoff between making heavy changes to the reused code and completely rewriting a new code. There is no doubt that the code reuse ratio will be higher than our numbers if we take into account the clone kinds of Type III and Type IV. For more accurate detection in the future, we intend to use clone detection techniques that can recognize type-III and type-IV clone types.

We conduct code reuse analysis on code changes in commits of GitHub projects and Stack Overflow, rather than using the complete code for reuse analysis. In fact, some programmers reuse code during the development process and do not always submit it to GitHub projects, but develop locally. In this case, we cannot obtain relevant data. In addition, we use a token-based clone detection tool (i.e. CCFinder), which is more stringent for clone detection results, so the actual code reuse ratio will be higher than 6.32%. Moreover, we performed the Cox Stuart trend test (Cox and Stuart, 1955) in RQ1 to verify that the proportion of code reuse will have a significant upward trend in the future, which further verifies that the phenomenon of code reuse will become common.

7. Conclusion & future work

This paper aims to comprehensively analyze the code reuse between Stack Overflow (SO, the world's largest programmer Q&A site) and GitHub (the most popular open-source project site). Different from some previous works, we focus on the analysis of code reuse in the process of software project evolution (i.e., the code reuse of code snippets at the commit granularity of open-source projects and code snippets on SO). We conduct a large-scale empirical study. The open-source Java project code dataset we construct contains 793 projects which include 342,148 modified code snippets. In addition, we build the SO code dataset which includes 1,355,617 posts. We employ CCFinder as our code clone detection tool. Some of the significant and interesting findings of our investigation are as follows:

Our empirical research analysis shows that code reuse between SO and GitHub is ubiquitous, and reveals some important conclusions. Our study results provide multiple practical insights for different stakeholders: researchers, developers, SO platform. In the future, we would like to extend our analysis objects to more Q&A website platforms and more open-source projects using other programming languages.

CRediT authorship contribution statement

Xiangping Chen: Methodology, Validation, Writing – original draft. **Furen Xu:** Data curation, Software, Writing – review & editing. **Yuan Huang:** Methodology, Resources, Writing – review & editing. **Xiaocong Zhou:** Project administration, Writing – review & editing. **Zibin Zheng:** Investigation, Validation, Writing – review & editing.

Declaration of competing interest

(1) This paper aims to comprehensively analyze the code reuse between Stack Overflow (SO, the world's largest programmer Q&A site) and GitHub (the most popular open-source project site).

(2) We focus on the analysis of code reuse in the process of software project evolution (i.e., the code reuse of code snippets at the commit granularity of open-source projects and code snippets on SO).

(3) The Open-source java project code dataset we construct contains 793 projects which include 342,148 modified code snippets. In addition, we build the SO code dataset which includes 1,355,617 posts.

(4) Our empirical research analysis shows that code reuse between SO and GitHub is ubiquitous, and reveals some important conclusions. Our study results provide multiple practical insights for different stakeholders: researchers, developers, SO platform.

Data availability

I have shared the link to my data/code in the paper.

Acknowledgments

The research is supported by National Key R&D Program of China (No. 2023YFB2703600), the National Natural Science Foundation of China (62372492), the Natural Science Foundation of Guangdong Province (2023A1515010746, 2023A1515011474).

References

Abdalkareem, R., Shihab, E., Rilling, J., 2017. On code reuse from stackoverflow: An exploratory study on android apps. Inf. Softw. Technol. 88, 148–158.

Ahmad, A., Feng, C., Li, K., Asim, S.M., Sun, T., 2019. Toward empirically investigating non-functional requirements of iOS developers on stack overflow. IEEE Access 7, 61145–61169.

An, L., Mlouki, O., Khomh, F., Antoniol, G., 2017. Stack overflow: a code laundering platform? In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, pp. 283–293.

Baltes, S., Kiefer, R., Diehl, S., 2017. Attribution required: Stack overflow code snippets in GitHub projects. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, pp. 161–163.

Baltes, S., Treude, C., 2020. Code duplication on stack overflow. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). IEEE, pp. 13–16.

Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L., 1998. Clone detection using abstract syntax trees. In: Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272). IEEE, pp. 368–377.

Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M., Klemmer, S.R., 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 1589–1598.

Chen, M., Fischer, F., Meng, N., Wang, X., Grossklags, J., 2019. How reliable is the crowdsourced knowledge of security implementation? In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp. 536–547.

Cordy, J.R., Roy, C.K., 2011. The NiCad clone detector. In: 2011 IEEE 19th International Conference on Program Comprehension. IEEE, pp. 219–220.

Cox, D.R., Stuart, A., 1955. Some quick sign tests for trend in location and dispersion. Biometrika 42 (1/2), 80–95.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186. http://dx.doi.org/10.18653/v1/N19-1423, URL https://aclanthology.org/N19-1423.

Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., Fahl, S., 2017. Stack overflow considered harmful? the impact of copy&paste on android application security. In: 2017 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 121–136.

Fluri, B., Wursch, M., PInzger, M., Gall, H., 2007. Change distilling: Tree differencing for fine-grained source code change extraction. IEEE Trans. Softw. Eng. 33 (11), 725–743.

Frakes, W.B., Kang, K., 2005. Software reuse research: Status and future. IEEE Trans. Softw. Eng. 31 (7), 529–536.

Gallardo-Valencia, R.E., Sim, S.E., 2009. Internet-scale code search. In: 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation. IEEE, pp. 49–52.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., et al., 2020. Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366.

Hoang, T., Kang, H.J., Lo, D., Lawall, J., 2020. Cc2vec: Distributed representations of code changes. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 518–529.

Hotelling, H., 1933. Analysis of a complex of statistical variables into principal components. J. Educ. Psychol. 24 (6), 417.

Hsieh, G., Kraut, R.E., Hudson, S.E., 2010. Why pay? Exploring how financial incentives are used for question & answer. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 305–314.

Jan, S.T., Wang, C., Zhang, Q., Wang, G., 2017. Analyzing payment based question and answering service. CoRR.

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

Komondoor, R., Horwitz, S., 2001. Using slicing to identify duplication in source code. In: International Static Analysis Symposium. Springer, pp. 40–56.

Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X., 2015. Learning entity and relation embeddings for knowledge graph completion. In: Twenty-Ninth AAAI Conference on Artificial Intelligence.

Liu, J., Chang, W.-C., Wu, Y., Yang, Y., 2017. Deep learning for extreme multi-label text classification. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 115–124.

Lopes, C.V., Maj, P., Martins, P., Saini, V., Yang, D., Zitny, J., Sajnani, H., Vitek, J., 2017. Déjàvu: a map of code duplicates on GitHub. In: Proceedings of the ACM on Programming Languages, Vol. 1. pp. 1–28.

Lotter, A., Licorish, S.A., Savarimuthu, B.T.R., Meldrum, S., 2018. Code reuse in stack overflow and popular open source java projects. In: 2018 25th Australasian Software Engineering Conference (ASWEC). IEEE, pp. 141–150.

Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., Hartmann, B., 2011. Design lessons from the fastest q&a site in the west. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2857–2866.

Manes, S.S., Baysal, O., 2019. How often and what StackOverflow posts do developers reference in their GitHub projects? In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, pp. 235–239.

Meng, N., Nagy, S., Yao, D., Zhuang, W., Argoty, G.A., 2018. Secure coding practices in java: Challenges and vulnerabilities. In: Proceedings of the 40th International Conference on Software Engineering. pp. 372–383.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Ni, C., Wang, W., Yang, K., Xia, X., Liu, K., Lo, D., 2022. The best of both worlds: integrating semantic features with expert features for defect prediction and localization. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 672–683.

Nishi, M.A., Ciborowska, A., Damevski, K., 2019. Characterizing duplicate code snippets between stack overflow and tutorials. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, pp. 240–244.

Ragkhitwetsagul, C., Krinke, J., Paixao, M., Bianco, G., Oliveto, R., 2019. Toxic code snippets on stack overflow. IEEE Trans. Softw. Eng..

Rieger, M., Ducasse, S., Lanza, M., 2004. Insights into system-wide code duplication. In: 11th Working Conference on Reverse Engineering. IEEE, pp. 100–109.

Röder, M., Both, A., Hinneburg, A., 2015. Exploring the space of topic coherence measures. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining. pp. 399–408.

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

Sadowski, C., Stolee, K.T., Elbaum, S., 2015. How developers search for code: a case study. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 191–201.

Sajnani, H., Saini, V., Svajlenko, J., Roy, C.K., Lopes, C.V., 2016. Sourcerercc: Scaling code clone detection to big-code. In: Proceedings of the 38th International Conference on Software Engineering. pp. 1157–1168.

Scoccia, G.L., Migliarini, P., Autili, M., 2021. Challenges in developing desktop web apps: a study of stack overflow and GitHub. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, pp. 271–282.

Shen, T., Zhou, T., Long, G., Jiang, J., Pan, S., Zhang, C., 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32.

Tairas, R., Gray, J., 2006. Phoenix-based clone detection using suffix trees. In: Proceedings of the 44th Annual Southeast Regional Conference. pp. 679–684.

Tóth, L., Nagy, B., Gyimóthy, T., Vidács, L., 2020. Why will my question be closed? nlp-based pre-submission predictions of question closing reasons on stack overflow. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). IEEE, pp. 45–48.

Vasilescu, B., Filkov, V., Serebrenik, A., 2013. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: 2013 International Conference on Social Computing. IEEE, pp. 188–195.

Wang, J., Dong, Y., 2020. Measurement of text similarity: a survey. Information 11 (9), 421.

Wu, D., Jing, X.-Y., Zhang, H., Feng, Y., Chen, H., Zhou, Y., Xu, B., 2023. Retrieving API knowledge from tutorials and stack overflow based on natural language queries. ACM Trans. Softw. Eng. Methodol. 32 (5), 1–36.

Wu, Y., Wang, S., Bezemer, C.-P., Inoue, K., 2019. How do developers utilize source code from stack overflow? Empir. Softw. Eng. 24 (2), 637–673.

Yang, D., Hussain, A., Lopes, C.V., 2016. From query to usable code: an analysis of stack overflow code snippets. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, pp. 391–401.

Yang, D., Martins, P., Saini, V., Lopes, C., 2017. Stack overflow in github: any snippets there? In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, pp. 280–290.

Yu, H., Lam, W., Chen, L., Li, G., Xie, T., Wang, Q., 2019. Neural detection of semantic code clones via tree-based convolution. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC). IEEE.

Zhang, M., Cui, Z., Neumann, M., Chen, Y., 2018. An end-to-end deep learning architecture for graph classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32.

Zhang, T., Song, M., Pinedo, J., Kim, M., 2015. Interactive code review for systematic changes. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. IEEE, pp. 111–122.

Zhang, H., Wang, S., Chen, T.-H.P., Zou, Y., Hassan, A.E., 2019a. An empirical study of obsolete answers on Stack Overflow. IEEE Trans. Softw. Eng..

Zhang, T., Yang, D., Lopes, C., Kim, M., 2019b. Analyzing and supporting adaptation of online code examples. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp. 316–327.

Zibran, M.F., Saha, R.K., Asaduzzaman, M., Roy, C.K., 2011. Analyzing and forecasting near-miss clones in evolving software: An empirical study. In: 2011 16th IEEE International Conference on Engineering of Complex Computer Systems. IEEE, pp. 295–304.

Zou, Y., Ban, B., Xue, Y., Xu, Y., 2020. CCGraph: a PDG-based code clone detector with approximate graph matching. In: 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 931–942.