How Do Developers Reuse StackOverflow Answers in Their GitHub Projects?

Juntong Chen
Virginia Tech
USA
j6xjun@gmail.com

Yan Zhao
Eastern Michigan University
USA
yanzhao@vt.edu

Na Meng
Virginia Tech
USA
nm8247@vt.edu

ABSTRACT

StackOverflow (SO) is a widely used question-and-answer (Q&A) website for software developers and computer scientists. GitHub is an online development platform used for storing, tracking, and collaborating on software projects. Prior work relates the information mined from both platforms without carefully inspecting the answer-reuse practices. For this paper, we did an empirical study by mining the SO answers reused by Java projects available on GitHub. We created a hybrid approach of clone detection, keyword-based search, and manual inspection, to identify the answer(s) actually used by developers. Based on those answers, we studied topics of the discussion threads, answer characteristics (e.g., scores, ages, code lengths, and text lengths), and developers' reuse practices.

We observed that most reused answers offer programs to implement specific coding tasks. Among all analyzed SO discussion threads, the reused answers often have higher scores, older ages, longer code, and longer text than unused answers. In only 9% of scenarios (40/430), developers fully copied answer code for reuse. In the remaining scenarios, they reused partial code or created brand new code from scratch. Our study characterized 130 SO discussion threads referred to by Java developers in 357 GitHub projects. Our observations can guide SO answerers to provide better answers, and shed lights on future human-centric research that creates better tools to help with code reuse.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → *Development frameworks and environments.*

KEYWORDS

Empirical, StackOverflow, GitHub, answer reuse, clone detection

ACM Reference Format:
Juntong Chen, Yan Zhao, and Na Meng. 2024. How Do Developers Reuse StackOverflow Answers in Their GitHub Projects?. In *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3691621.3694945

ASEW '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1249-4/24/10…$15.00
https://doi.org/10.1145/3691621.3694945

1 INTRODUCTION

Each month, about 50 million people visit StackOverflow (SO) to learn, share, and build their careers. Industry estimates suggest that 20–25 million of these people are professional developers and univerity-level students [28]. In February 2020, a survey with nearly 65,000 developers shows that when stuck on a coding problem, 90% of respondents visited SO. The fact implies that SO plays an important role in shaping the art and practices of software today.

Studies were conducted to characterize the crowdsourced knowledge available on SO, or to relate the mined knowledge from SO and from GitHub [30]. For example, Nasehi et al. [46] manually inspected 163 unique SO posts with well-received answers; they found that the explanations accompanying examples are as important as the code examples themselves. Vasilescu et al. [50] identified GitHub developers active on SO, to study their activities on both platforms. They observed that GitHub committers ask fewer questions and provide more answers than others. Manes and Baysal [41] mined the SOTorrent dataset; they found that on average, developers make 45 references to SO posts in their GitHub projects, with the highest number of references made in JavaScript code.

However, limited knowledge is available concerning how developers leverage the crowdsourced knowledge available on SO, and what kind of SO posts got referenced by GitHub projects. Such knowledge can guide answerers to provide better answers, assist questioners to better compare answers, and shed light on new tools that recommend customized coding solutions to developers based on SO answers. For this paper, we did an empirical study to explore the following research questions to complement prior work:

RQ1 *What kinds of SO discussion threads are referenced by GitHub projects?* When certain discussion threads have URLs cited by GitHub projects, we aimed to characterize the discussion topics. Those topics can reflect developers' focus when seeking for coding assistance; thus, they can guide researchers, tool builders, and SO answerers to better help developers.

RQ2 *What are the characteristics of reused SO answers?* When an SO discussion thread has multiple alternative answers, we aimed to characterize the factors (e.g., scores and code lengths) that may influence developers' decisions on choosing some answers over the others. Such characteristics can guide SO answerers to provide answers with higher quality, and to earn reputation more effectively.

RQ3 *How are SO answers reused in GitHub projects?* When referring to SO answers, developers may reuse some or all exemplar code, revise the code as needed, or create totally new code based on the insights. By characterizing the answer-reuse practices of GitHub developers, we intended to study how SO answers help shape software products.

To investigate the research questions mentioned above, we first used a fully managed enterprise data warehouse Google BigQuery [32], to crawl for Java files in GitHub projects that reference any SO discussion thread (i.e., URL), and to download 30,000 answer posts contained by all those referenced SO discussion threads. We chose Java because it is widely used and we are more familiar with the language; such a familarity enables us to manually analyze SO posts and GitHub code with high confidence. An **SO discussion thread** typically has a question post and one or more answer posts. As each post has a URL, developers may reference a discussion thread via the URL of question and/or any answer in that thread. For each of the 30,000 downloaded answers, we extracted exemplar code, text (i.e., code + natural-language explanation), and descriptive metadata (e.g., scores and creation timestamps).

Based on the crawled data, we applied a novel hybrid approach to decide which answer(s) in a thread were actually used by developers. This approach has three steps. In Step 1, it uses a clone detection tool—PMD [31]—to find any answer that has code similar to the Java code on GitHub. Our insight is that if developers reference an SO thread in their code, it is likely that they create certain duplicates of the answer code from that thread. Step 2: as PMD may be insufficient to locate all answer reuses, we also extracted the IDs of all answer posts in our dataset. We searched for Java files that explicitly mention any of these IDs, considering them as indicators for answer reuse. Step 3: we manually inspected the GitHub→SO reference links revealed by Steps 1 and 2, to refine the dataset.

Our study revealed in total 130 SO discussion threads referenced by 407 Java files, which belong to 357 GitHub projects. Most of the threads (i.e., 78%) are about solutions to coding problems: question askers describe their software requirements and answer providers offer programs to satisfy those requirements (RQ1). The studied 130 threads have in total 2,323 answers. Using our hybrid approach, we found 186 answers reused by GitHub codebases. When ranking all answers in each thread, we observed that the reused answers have statistically higher scores, older ages, larger code, and more text (RQ2). Among the 430 scenarios of answer reuse, we found fully identical code snippets between GitHub and SO in only 40 scenarios. Most developers reused SO answers by applying changes to the suggested code, or creating code from scratch based on the ideas described by those answers.

In this paper, we made the following research contributions:

- We created a new approach to identify SO answers reused in open-source Java projects. By combining clone detection, keyword-based search, and manual inspection, this approach was able to identify reused answers effectively.
- We did a novel empirical study to characterize the SO answers reused by open-source projects on GitHub. Our study characterized the reused answers from unique angles like the discussion topics, the content, post ages, and post scores. No prior studies considered these aspects.
- We defined a taxonomy of patterns to describe developers' answer-reuse practices. No prior work categorizes developers' reuse practices in such a comprehensive way.

In the following sections, we will introduce the background (Section 2), our data collection process (Section 3), and experiments (Section 4). We open-sourced our material (i.e., program and data) at https://figshare.com/articles/dataset/So-gitexperiment/20425839.

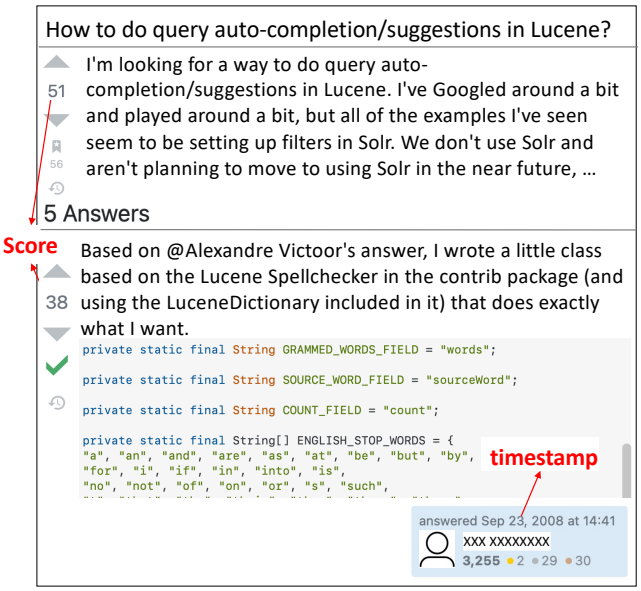

Figure 1: An exemplar SO discussion thread that contains one question post and multiple answer posts [7]

```
/**
  * Preliminary autocompete implementation based on
  * http://stackoverflow.com/questions/120180/how-to-do-query-
  * auto-completion-suggestions-in-lucene
  *
  * @author craig.willis@unc.edu
  * /
public final class Autocomplete {
  /* Lucene index field to store concept ID */
  private static final String ID_FIELD = "id";

  /* Lucene index field to store the grammed words */
  private static final String GRAMMED_WORDS_FIELD = "words";

  /* Lucene index field to store the full source word */
  private static final String SOURCE_WORD_FIELD = "sourceWord";

  /* Lucene index field to store the word count */
  private static final String COUNT_FIELD = "count";
  …
```

The cited SO link (corresponding to Figure 1)

The code derived from SO

Figure 2: An exemplar Java file on GitHub that cites an SO thread and reuses some of the answer code [21]

2 BACKGROUND

This section introduces the content of a typical StackOverflow discussion thread; it also shows how a GitHub project references a discussion thread and reuses one of the answers.

As shown in Figure 1, an SO discussion thread has one **question** and zero, one, or multiple **answers** to that question. After a user posts a question or answer, other users can vote for or against that post. The **score** is decided by the up- and down- votes received by a post. Each post has a **timestamp** to show when it was created.

StackOverflow assigns each question or answer a unique **ID number**, while the question ID is also treated as the **unique ID of its discussion thread**. A typical *SO question link* has the format `https://stackoverflow.com/questions/ question_ID/question_title`, while an *answer URL* often has the format `https://stackoverflow.com/questions/ question_ID/question_title/answer _ID#answer_ID`.

Note that the answer URL contains IDs of both the question and answer. The question and answer links mentioned above can be used to retrieve their corresponding discussion threads from StackOverflow. For instance, developers can retrieve the discussion thread shown in Figure 1 via the question URL

"https://stackoverflow.com/questions/120180/how-to-do-query-auto-completion-suggestions-in-lucene",
or the URL of any answer it contains, such as

"https://stackoverflow.com/questions/120180/how-to-do-query-auto-completion-suggestions-in-lucene/121456#121456".
Here "120180" is the question's ID; "121456" is the answer's ID. In the datasets related to StackOverflow (e.g., Google BigQuery [32] and StackOverflow data dump [29]), a question/thread ID is often used as the **parent ID** of some answers, showing that those answers respond to the given question.

Figure 2 shows a Java file on GitHub that references/cites the above-mentioned SO thread via its question link. Comparing the code content of Java file and the first answer in the thread, we can find common code that indicates developers' answer-reuse practice. Our research focuses on (1) the Java files from GitHub that explicitly reference URLs of SO questions or answers, and (2) the SO threads retrieved via those URLs.

3 DATA COLLECTION

To create a dataset of SO answers reused by GitHub projects, in 2021, we used Google BigQuery [32] to crawl for SO links that are mentioned by Java files on GitHub (Section 3.1), and applied a new hybrid approach to identify reused SO answers (Section 3.2).

3.1 Data Crawling

Google BigQuery is a serverless data warehouse that enables scalable data analysis. It is a Platform as a Service (PaaS) that supports SQL queries. It hosts datasets like GitHub project data and SO post information, and supports data queries across multiple datasets. Our research focuses on (1) the Java files on GitHub citing any SO links and (2) the SO posts related to cited links. Therefore, BigQuery satisfies our need.

As shown in Listing 1, we defined an SQL query for BigQuery. The query performs three major tasks. First, it retrieves Java files from GitHub that cite SO links and records those links (see lines 5–9). Second, treating each recorded link as the URL of a discussion thread, it retrieves all answer posts from SO belonging to that thread (see lines 1–10). Third, it orders retrieved answers based on the discussion-thread ID (i.e., `parent_id`), and limits the total number of retrieved answers to 30,000 (see lines 11–12). We set the upper bound to 30,000 for two reasons. First, our free user account with BigQuery limits the number of SQL queries to execute each month, and the amount of data processed by each query. Second, our later data analysis involves manual inspection. Even if BigQuery can return all records satisfying the query, our manual inspection is unscalable and can only analyze a subset. After trying different limit settings, we found 30,000 to be a reasonable number to handle.

Listing 1: Our SQL query to crawl GitHub and StackOverflow

```
1 SELECT a.id, title, body, content, parent_id, favorite_count favs,
        view_count views,  score, accepted_answer_id, post_type_id,
        sample_repo_name, sample_path
2 FROM
3   'bigquery-public-data.stackoverflow.posts_answers' a
4 INNER JOIN (
5   SELECT
6     CAST(REGEXP_EXTRACT(content, r'stackoverflow.com/questions
          /([0-9]+)/') AS INT64) id, sample_repo_name, sample_path,
          content
7   FROM `fh-bigquery.github_extracts.contents_java`
8   WHERE
9     content LIKE '%stackoverflow.com/questions/%') b
10 ON a.parent_id=b.id
11 ORDER BY parent_id
12 LIMIT 30000
```

Among the records retrieved from Google BigQuery, we realized that no creation date of posts is included, and the `body` field does not always include the complete content of answer posts. As our later analysis heavily depends on the availability of posts' creation date and answer content, we conducted additional crawling on StackOverflow data dump [29] to acquire that information. In summary, our data crawling obtained 30,000 answer posts. These posts contain duplicates because when the same question ID is referenced multiple times, all of its answers are retrieved again and again. After removing duplicates, we identified 322 unique SO discussion threads to cover all crawled answers. These threads are referenced by 1,254 unique Java files from 1,063 GitHub projects.

3.2 Detection of Answer Reuse

As mentioned in Section 2, a Java file may reference an SO question and/or answer link. If an answer link is referenced, detecting answer reuse is simple: we only need to compare the Java file with cited answer to comprehend the reuse practice. However, if no answer is referenced and there is only one question link, detecting answer reuse is harder. As a question often corresponds to multiple answers, it is not always easy to decide which answer is more relevant. We believe that when developers reuse SO answers, they may reuse the code snippets mentioned in those answers; thus, we can find reused answers by detecting code clones (i.e., similar code) between each answer and the Java file. Based on this insight, We created a three-step hybrid approach to detect reused answers.

3.2.1 Step 1: Detecting reused answers via clone detection. When a Java file f references an SO question link, we located all answers to that question. To facilitate discussion, we use A to denote the located answer set. From each answer $a_i \in A$, we extracted all code snippets and stored them in a single Java file j_i. We denote all Java files synthesized in this way with J. Afterwards, we applied PMD Copy/Paste Detector (CPD) [31]—a clone detection tool—to f and J, in order to find code clones between the Java file and answer code. We chose PMD because it is publicly available, easy to use, and often applicable given two files for comparison. When comparing Java programs, PMD treats source code as plain text and adopts a string-matching algorithm [27] to compare the hash values of strings. It does not matter if some synthesized Java files are incompilable or have lexical/syntactic errors, because PMD does not observe program syntax or semantics anyway.

PMD has a parameter `minimumTokenCount` to specify the minimum number of tokens contained by a reported clone. This parameter controls trade-offs between the effectiveness of clone detection and the usefulness of detected clones. Namely, if the parameter value is too small (e.g., 2), PMD can find a lot of clones; however, many of the clones may share as few as two tokens and seem to be accidental code overlap instead of meaningful code reuse. Meanwhile, if the

Table 1: Number of clones PMD reported when `minimumTokenCount` was set differently

`minimumTokenCount =`	5	10	15	20	25
# of clone pairs examined	30	22	17	12	10
# of true positives	12	12	12	12	10
Precision	40%	55%	71%	100%	100%

value is too large (e.g., 500), PMD may only report few clones while each reported clone pair shows strong evidence of code reuse.

To properly configure `minimumTokenCount`, we conducted a preliminary study by experimenting PMD with different parameter values. Specifically, we randomly picked 30 pairs of $\langle f, a_i \rangle$. Here, f represents a Java file from GitHub referencing a discussion thread, and a_i represents a Java file synthesized from an answer of that thread. We tuned `minimumTokenCount` from 5 to 25, with 5 increments. For each parameter setting, we applied PMD to the 30 $\langle f, a_i \rangle$ file pairs for clone detection. Then we manually inspected results to see which setting achieves a better trade-off between the number and quality of reported clones. For simplicity, if multiple clones are reported for a given file pair, we checked only the first clone pair to assess whether the code has meaningful similarity or just random content overlap. With "**random/accidental content overlap**", we mean a few common tokens shared between two totally different statements. In this experiment, for each setting, we examined at most 30 clone pairs when PMD reported clones for all file pairs.

As shown in Table 1, when `minimumTokenCount` increases, the total number of clone pairs examined decreases. This is because when more common tokens are required between clones, PMD reported clones for fewer file pairs. Among the manually checked clone pairs, we identified 12 true clone pairs when `minimumTokenCount` was 5, 10, 15, or 20. However, only 10 true clone pairs were identified when the parameter was 25. Finally, we computed the precision rate for PMD by dividing the number of true positives/clones with the total number of clone pairs examined. When the parameter increases, the precision rate increases or remains. Considering both the total number of true clones revealed and precision rate, we found `minimumTokenCount=20` to be a better setting than the others.

We did another experiment to further tune the parameter from 17 to 20 tokens, with 1 increments. We wanted to explore a setting that outperforms 20 by achieving a better trade-off between the number of true positives and the precision rate. We found 18, 19, and 20 to produce equally good experiment results. Therefore, by default, we set `minimumTokenCount=18`, to retrieve as many true positives as possible while ensuring high precision. With this setting, PMD reported clones for 1,526 $\langle f, a_i \rangle$ pairs in our dataset, which pairs are considered as candidates to show answer reuse.

3.2.2 Step 2: Detecting reused answers via keyword-based search. When a Java file f references an SO answer link, we consider the answer to be potentially reused by f. Based on our experience, although Step 1 is effective in finding candidates, it can miss reused answers in three kinds of scenarios. First, there is no code mentioned in an answer, but developers reused that answer by digesting the idea and writing code accordingly. Second, developers got inspired by code mentioned in an answer, but wrote totally different code. Third, due to some unknown implementation issues, PMD fails to identify code clones between certain similar code.

To find the reused answers potentially missed by PMD, we decided to rely on SO answer links explicitly referenced by Java files. Specifically, we extracted the IDs of 30,000 retrieved answers, and used them as keywords to search among the 1,254 Java files we crawled (see Section 3.1). If a Java file contains any of the answer IDs, the file is considered to potentially reuse that answer. With this approach, we found 89 Java files citing the answers under analysis.

3.2.3 Step 3: Manual inspection to refine detected answer reuses. Among the $\langle f, a_i \rangle$ pairs that PMD reported to have clones, not every pair implies an actual answer reuse. For instance, if the latest editing date of a Java file f is in 2015 but f has code similarity with an SO answer a_i posted in 2021, f can never reuse a_i as the answer did not exist in 2015. To identify actual answer reuses, we used the following four criteria to remove unpromising pairs:

- **Date Comparison:** If $\langle f, a_i \rangle$ has the creation date of a_i later than the latest editing date of f, the pair does not imply answer reuse as developers could not refer to a nonexistent answer when editing their file.
- **Similarity Comparison:** If (1) f is reported to have clones with several answers from the same thread and (2) those answers share code, we consider the answer that has the highest code similarity with f as the reused answer. We believe that when reusing answer code, developers are more likely to focus on one answer instead of reusing multiple answers simultaneously.
- **ID Comparison:** If f explicitly cites an answer link and gets reported to have clones with several answers from the same thread, we consider the cited answer to be reused and treat all other answers as false positives. This is because developers are likely to reference the answers that help them most, giving credits to the reused answers.
- **Availability Checking:** If f is not available because developers recently removed that file or the whole project, we cannot compare the file with any candidate answers it may reuse. Thus, we removed all pairs containing f.

Our manual analysis of clone pairs was very time-consuming and error-prone. To reduce human errors, two authors separately checked reported file pairs, and then held meetings to go over the list. They compared their manual inspection results; whenever the results diverged, they discussed comprehensively to reach a consensus.

After identifying answer reuses based on PMD results, we further examined the 89 Java files found via keyword-based search in Step 2. We used two major criteria to decide whether a file's answer reference should be considered as answer reuse:

- **Duplicated Entries:** Suppose that f explicitly cites a_i, while the pair $\langle f, a_i \rangle$ is already included into our dataset of answer reuse due to clone detection. In such scenarios, the Java file's answer reference is not added redundantly.
- **Availability Checking:** If f is unavailable because developers recently removed it or the whole project, we cannot compare the file with any answer it references. Thus, the Java file's answer reference is not added to our dataset.

Two authors followed the criteria mentioned above to separately examine the reported references. They then compared and discussed results until reaching a consensus.

By applying the hybrid approach mentioned above, we got a refined dataset to include 130 discussion threads, which contain 2,323

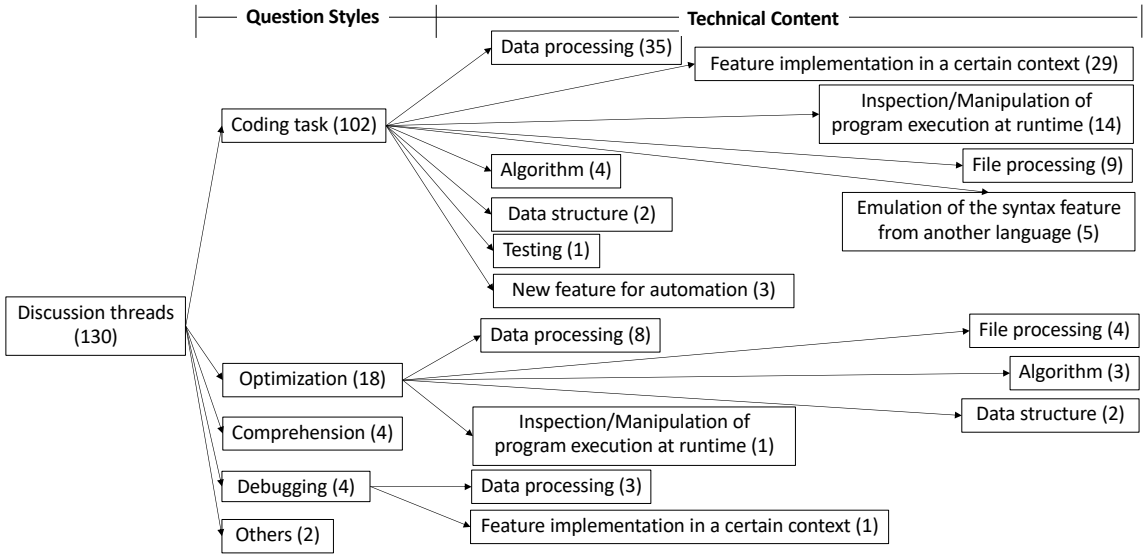

Figure 3: Our taxonomy of SO threads based on the discussion topics

answers in total. Among those answers, 186 answers were found to be reused, and 2,137 answers were unused. The 186 answers were reused 430 times; 407 Java files from 357 unique projects reused those answers. Our later experiments (see Section 4) will focus on this refined dataset.

4 EXPERIMENTS

This section presents our investigation for the three research questions RQ1–RQ3. For each RQ, we will first introduce the study method and then describe our experiment results.

4.1 RQ1: Topics of Discussion Threads

Study Method. RQ1 captures the focus of developers when they reused answers. We performed a lightweight open coding-like process [49] to categorize the topics discussed in 130 threads. Specifically, while the last author (a professor with SE expertise) manually inspected all 130 questions in threads, she extracted or summarized keywords to characterize each question in terms of the question style and technical content. Then she identified the commonality between questions based on keywords, defined categories accordingly, and revisited all questions to decide whether the categories were comprehensive enough to cover all questions. The author defined and refined categories iteratively, until each question was mapped to a category and the category list was stable. Next, the first author manually checked the categorization results, and discussed with the last author if any classification seemed problematic.

Results. Figure 3 presents our classification results of SO threads. In terms of the question style, we identified 5 categories in the 130 discussion threads: coding task, optimization, comprehension, debugging, and others. *Coding task* means that askers describe software requirements (e.g., how to validate an XML file against an XSD file [9]) and seek for code solutions, while answerers offer code to implement those requirements. *Optimization* means that askers provide initial programs satisfying certain requirements, looking for better programs that have either easier implementation, lower runtime overheads, or less platform-specific dependency [13]. An exemplar thread of this category is about the most efficient way to implement an integer-based power function pow(int, int) [13]. Optimization is different from the coding-task category, as askers provide initial code implementation.

Comprehension is about clarification or comparison of concepts or terms. For instance, an SO thread compares two Java APIs: StringBuilder versus StringBuffer [2]. *Debugging* means that askers present their erroneous programs and seek for debugging advices. An example of this category is about a strange OutOfMemory issue while loading an image to a Bitmap object [20]. *Others* captures the miscellaneous questions not covered by any category mentioned above. Specifically, one of the two questions is about SQLite command usage [17], and the other focuses on issues that developers should consider when overriding `equals(...)` and `hashCode()` [15].

Among the five major categories, we noticed that coding task dominates the discussion threads. It covers 102 of the 130 threads. The second biggest category is optimization, which covers 18 threads. These observations imply that when developers reuse SO answers, they often focus on the answers that offer complete (or even optimized) programs to satisfy certain requirements. Surprisingly, comprehension and debugging separately cover only four threads. It means that although StackOverflow provides a platform for developers to discuss technical concepts or software bugs, developers rarely reused the answers related to concept comprehension or bug fixes. This may be because comprehension-related discussion is too general or abstract for developers to adopt in code, while debugging-related discussion is too specific or concrete for developers to reuse in their diverse circumstances.

Finding 1: *92% (120/130) of discussion threads are about coding tasks or optimizations. The reused answers often provide complete (or even optimized) coding solution given software requirements.*

In terms of the technical content, we identified 9 topics in the 130 threads (see Figure 3). *Data processing* is about how to generate, handle, or transform data. For instance, a thread discusses how to do 3DES data encryption/decryption [6]. Similarly, *File processing* is about how to handle or compare files. An exemplar thread is about

optimized ways of counting lines in a file [19]. Concerning *Feature implementation in a certain context*, askers describe their software requirements in specialized circumstances (e.g., when using certain Java libraries), and seek for suggestions applicable to those circumstances. For instance, a thread discusses how to make a list with checkboxes in Java Swing [5]. With *Inspection/Manipulation of program execution at runtime*, we refer to discussions on how to programmatically inspect or manipulate execution status/environments at runtime, such as programmatically checking CPU and memory usage [4].

Algorithm focuses on the design and implementation of algorithms or mathematical computation, such as calculating the distance between two latitude-longitude points [1]. *Emulation of the syntax feature from another language* focuses on how to emulate certain syntax features offered by languages other than Java, such as the as-operator of C# [8]. *Data structure* covers discussions on defining customized data structures (e.g., an LRU cache [10]). *Testing* is about defining test cases. For instance, one thread discusses how to test methods that call System.exit() [11]. *New feature for automation* describes the threads that discuss rare feature implementation, which is not covered by any of the topics mentioned above. For instance, a thread discusses how to play sound in Java [33].

Among the five categories mentioned above, the dominant category coding task covers all nine topics. In particular, within the 102 discussion threads, 35 threads focus on data processing, 29 threads discuss feature implementation in a certain context, and 14 threads are about the inspection or manipulation of program execution at runtime. The optimization category covers five topics: data processing, file processing, algorithm, data structure, and inspection/manipulation of program execution at runtime. The debugging category covers even fewer topics: data processing and feature implementation in a certain context, mainly because there are a lot fewer threads covered by this category.

Among the nine topics, the most popular four topics are: data processing, feature implementation in a certain context, inspection/-manipulation of program execution at runtime, and file processing. Each of the topics separately covers 46, 30, 15, and 13 threads.

Finding 2: *80% (104/130) of threads are on the topics of data processing, feature implementation in a certain context, inspection/-manipulation of program execution at runtime, and file processing.*

4.2 RQ2: Characteristics of Reused SO Answers

Study Method. RQ2 explores which of the following characteristics can effectively differentiate reused answers from unused ones: post scores, post ages, lengths of answer code, and lengths of the text (i.e., code + explanation). A challenge in comparing the two sets (reused vs. unused answers) is that we cannot naïvely compare the measured absolute values, because the measured values from different threads can vary a lot. For instance, one discussion thread has answer scores vary within [1, 5], while another thread has the scores in [10, 5000]. If we naïvely extract scores (or other values) and compare the distributions, it is almost impossible for us to see any meaningful difference between sets.

To overcome the challenge, we defined an approach to compare the two sets based on relative ranks of answers in each thread. Specifically, we wrote scripts to automatically sort the answers of each discussion thread based on four alternative metrics: post scores, post ages, code lengths, and text lengths. The post scores were directly extracted from the data crawled by BigQuery (see Section 3.1). The post ages were computed based on the date difference between each post's creation and our experiment. Both code lengths and text lengths were based on the complete answer content extracted from SO data dump. In particular, code length is the character count of each answer code, while the text length is the character count of each answer. In our experiment, the maximum and minimum text lengths are separately 24,367 and 29.

For each sorting task, we ranked answer posts in descending order of measured values, because we intended to explore whether reused answers were usually ranked differently from unused answers. Based on the ranking results, we computed a percentile rank (PR) [48] for each answer as below:

$$PR(v) = \frac{CF(v) - 0.5 \times F(v)}{N} \times 100 \quad (1)$$

PR is within [0, 100]. CF means *cumulative frequency*—the count of all values less than or equal to the value of interest v. F is the *frequency* of v. N is the total number of answers in the ranked list. For instance, if we have a ranked list {5, 5, 4, 3, 2, 1, 0}. Then $PR(5) = \frac{7-0.5\times2}{7} \times 100 = 88$. As another example, when posts are ranked in descending order of ages, the oldest and newest posts separately get the highest and lowest PR values.

By mapping concrete measurements to PR values, we compared the relative rank distributions of reused and unused answers within their separate discussion threads. In data analytics, violin plots visualize the distributions of numerical data [39]. They show not only summary statistics (e.g., median and interquartile range), but also the density of each variable. We used violin plots to visualize the PR distributions of reused answers, unused answers, and all answers of the 130 threads. We also conducted Mann-Whitney U test [43] to check whether the distributions of reused and unused answers present statistically significant differences.

Results. Figure 4 visualizes the PR distributions of three answer sets based on four measurements: score, age, code length, and text length. The *Reused* set clusters the 186 answers reused by 407 Java files. The *Unused* set clusters the remaining 2,137 answers included by the 130 threads. The *All* set includes all 2,323 answers.

As shown in Figure 4, reused answers usually have higher scores, older ages, larger code, and longer text than unused ones. In terms of scores (Figure 4 (a)), the PR values of reused answers have the mean as 79 and median as 87. The upper quartile (i.e., the value under which 75% of data points are found) is 94, and the lower quartile (i.e., the value under which 25% of data points are found) is 69. The data peak is near the upper quartile—94. On the other hand, the PR values of unused answers have the mean as 47 and median as 46. The plots for both unused and all answers have no obvious peak. One possible reason to explain our observations is that scores reflect the quality of answers. As developers often strive to reuse the answers with highest quality, it is unsurprising to see that reused answers often have much better scores than unused ones. Our observations imply that when developers hesitate to choose answers from a thread, they can pay more attention to the few with highest scores.

In terms of ages (Figure 4 (b)), the PR values of reused answers have the mean as 69 and median as 74. The PR values of unused answers have both the mean and median as 48. One possible reason

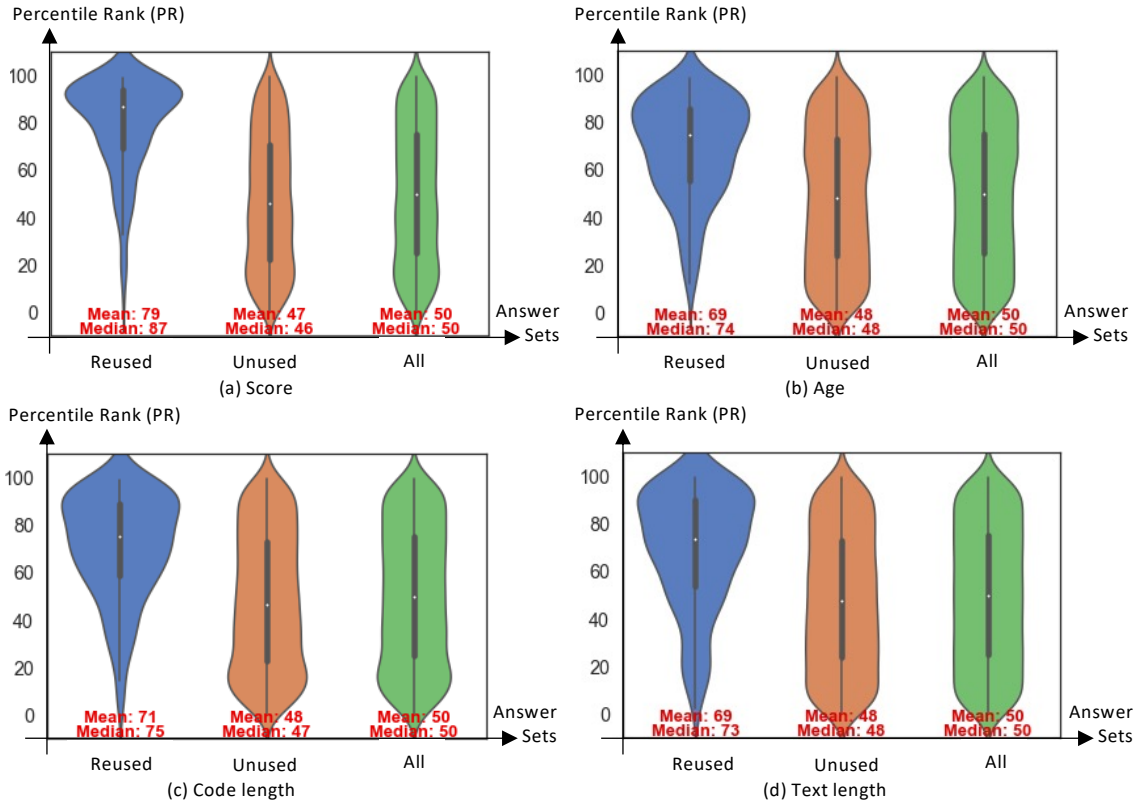

Figure 4: The PR-comparison among reused answers, unused answers, and all answers of the 130 threads

to explain the phenomena is our data-crawling process. Because we crawled GitHub projects to reveal answer-reuse practices, many of the practices reflected by source code were actually conducted months or years ago. Thus, the answers reused are typically older. In terms of code lengths and text lengths (Figure 4 (c) & Figure 4 (d)), we observed similar phenomena. In both graphs, reused answers have 73–75 medians, while unused answers have 47–48 medians. One possible reason to explain our observations is that the longer code or text an answer has, the easier it is for developers to digest the idea and reuse that answer. Our observations imply that when developers hesitate to choose answers for reuse, they can focus more on the answers with larger code or longer description.

The Mann-Whitney U test we performed shows $p << 0.05$ for all measurements, meaning that the reused and unused answers have statistically significant differences in terms of their PR ranks by score, age, code length, and text length.

Finding 3: *Compared with unused answers, reused ones have statistically higher scores, older ages, longer code, and longer text.*

4.3 RQ3: Categorization of Reuse Practices

Study Method. RQ3 investigates how developers reused SO answers in their software projects. We took a lightweight open coding-like process to categorize developers' reuse practices demonstrated in 357 GitHub projects. Specifically, 2 authors separately checked 430 cases (i.e., code locations) where SO answers were reused, and defined keywords to summarize the similarity between Java files and SO answers. Then they held a meeting to compare summaries for different kinds of cases, discussed the rationale behind those summaries, and brainstormed a set of categories that are exclusive of each other but sufficient to cover all observed cases. Next, the authors divided the 430 cases into 2 sets, and separately worked on a set to label the reuse practices based on the category set they agreed upon. Afterwards, they crosschecked the labels assigned by each other, and had discussions as needed to reach a consensus.

Results. Depending on the observed similarities between Java files and the SO answers they reused, as with prior work [51], we identified 5 major types (i.e., C1–C5) of answer-reuse practices. As shown in Table 2, C1–C3 involve copying and pasting code, while C4–C5 do not copy or paste any code. In particular, C1 means exact copy: developers fully copy and paste code without modifying anything [18, 22]. C2 means full copy with cosmetic modifications like identifier renaming and literal replacement [3, 25]. C3 means copy with non-cosmetic modifications [16, 26]. C4 means idea reuse: an answer illustrates coding ideas via code examples, algorithm explanation, or images [14, 24]; developers reuse the ideas without reusing any code. C5 means knowledge learning: an answer explains certain concepts or terms; developers digest the concepts/terms to independently work on their coding tasks [12, 23].

Because C4 and C5 have no code reuse, we recognized the reuse practices for both categories based on the answer IDs cited by Java files and manual inspection (Section 3.2). Namely, If a Java file cites an answer and implements the algorithm described by that answer, we considered the answer reuse as C4; otherwise, it is of C5. Because C1–C3 have code reuse, we identified the reuse practices for these categories mainly based on clone detection and manual inspection.

Table 2: The identified types of developers' answer-reuse practices among the 430 reuse scenarios

Idx		Name	Definition	Count	Percentage
C1		Exact copy	Developers copy and paste *all* code, without any modification.	40	9%
C2		Copy with cosmetic modification	Developers copy and paste *all* code, and apply minor modification without changing the program structure. The minor modifications are limited to updates to identifiers and literals.	94	22%
C3	C3.1	Reuse all with statement-level updates only	Developers copy and paste *all* code, and update content of some statements.	10	2%
	C3.2	Reuse all with structure changes only	Developers copy and paste *all* code; they modify the program structure by inserting, deleting, reordering, or combining statements.	13	3%
	C3.3	Reuse all with both statement-level updates and structure changes	Developers copy and paste *all* code; they update content of some statements; they also revise the program structure by inserting, deleting, reordering, or combining statements.	57	13%
	C3.4	Reuse most without change	Developers copy and paste *most* code, without applying any change.	13	3%
	C3.5	Reuse most with statement-level updates only	Developers copy and paste *most* code, and update content of some statements.	21	5%
	C3.6	Reuse most with structure changes only	Developers copy and paste *most* code; they modify the program structure by inserting, deleting, reordering, or combining statements.	12	3%
	C3.7	Reuse most with both statement-level updates and structural changes	Developers copy and paste *most* code; they update content of some statements; they also modify the program structure by inserting, deleting, reordering, or combining statements.	84	20%
	C3.8	Reuse some without change	Developers copy and paste *some* code, without applying any change.	3	1%
	C3.9	Reuse some with statement-level updates only	Developers copy and paste *some* code, and update content of certain statements.	19	4%
	C3.10	Reuse some with structure changes only	Developers copy and paste *some* code; they modify the program structure by inserting, deleting, reordering, or combining statements.	2	*0%
	C3.11	Reuse some with both statement-level updates and structural changes	Developers copy and paste *some* code; they update content of certain statements; they also modify the program structure by inserting, deleting, reordering, or combining statements.	38	9%
C4		Converting ideas	Developers do not copy or paste any code. Instead, they write code from scratch based on the ideas shown by the code example(s), algorithm description, or image(s) in an answer.	7	2%
C5		Learning knowledge	Developers do not copy or paste any code. Instead, they write code based on the concepts or term definitions described in an SO answer.	17	4%

* 0% is actually 0.47%. The table shows 0% because we rounded the measured value to the nearest integer.

As shown in Table 2, developers created exact copies for answer code in only 9% of cases; they modified and reused answer code in 85% of cases, and did not reuse any answer code in 6%. These numbers imply (1) developers seldom used answer code as is, but customized code before reuse; (2) when reusing answers, developers often reuse code. Thus, if answerers want to better influence software practitioners, they'd better provide code examples in answers. Among C1–C3, C3 is the largest category, covering 63% of scenarios. To better characterize developers' reuse practices in these scenarios, we defined 11 subcategories for C3 based on 3 criteria:

(1) **Did GitHub developers reuse all, most, or some of the answer code?** Given an answer, if a Java file has counterparts (i.e., similar or identical statements) for all statements in the answer code, the reuse level is "all". If a Java file has counterparts for more than 50% of the statements in answer code, we use "most"; otherwise, we use "some".
(2) **Did GitHub developers modify the content of any statement in the answer code?** If developers updated any statement in the answer code before reusing it, and if the updated code is similar to the original one, we use "yes" to mark the update. Otherwise, we use "no".
(3) **Did GitHub developers modify the program structure of answer code?** If developers added, deleted, reordered, or combined statements in the answer code, we use "yes" to mark the structure changes. Otherwise, we use "no". Structure change is orthogonal to statement update, as statement updates do not involve adding, deleting, reordering, or combining statements; they are about changes inside statements.

C3.3, C3.7, and C3.11 are the three most popular ones among the 11 subcategories. Cases in these subcategories applied both statement-level updates and structure changes to the reused code. Six subcategories involve structure changes: C3.2, C3.3, C3.6, C3.7, C3.10, and C3.11; they account for 48% of overall reuses. Six subcategories involve statement-level updates: C3.1, C3.3, C3.5, C3.7, C3.9, C3.11; they account for 53% in total. Our observations imply that when developers applied non-cosmetic changes, they were more likely to apply statement-level updates than structure changes.

Finding 4: *Most developers reused at least some of the answer code. Developers often revised the code-to-reuse by applying statement-level updates and/or structure changes.*

5 RELATED WORK

The related work includes studies on SO, and studies on the relationship between SO and GitHub.

5.1 Studies on StackOverflow

Researchers performed studies to characterize the crowd-sourced knowledge on StackOverflow [37, 38, 40, 44–46, 54]. Specifically, Movshovitz-Attias et al. [45] analyzed the SO reputation system to identify the participation patterns of high and low reputation users. Honsel et al. [40] first interviewed five developers to identify the nine myths they believed, and then analyzed SO data to check those myths. Gantayat et al. [38] studied the synergy between voting and acceptance of answers on SO, finding the accepted answers to be top-voted in 81% of threads. Some researchers examined SO threads for specialized domains. For instance, Zhang et al. [54] studied the code examples of API usage, to reveal answers with API misuses. Meng et al. [44], and Chen et al. [37] examined SO posts on Java security, to reveal developers' concerns on security implementation, technical challenges, or vulnerabilities in answer code.

Our study is related to all studies summarized above, but has a unique focus on the answer reuse by open-source Java projects available on GitHub. The study most related to our work was conducted by Nasehi et al. [46], who manually inspected 163 discussion threads, and identified several characteristics of well-received answers (i.e., answers with score 4 or above). Our data analysis complements Nasehi's study.

5.2 Studies on SO–GitHub Connections

Studies were conducted to investigate the relationship between data mined from StackOverflow, and the data from GitHub [34–36, 41, 42, 51–53]. Some studies associated users across platforms via common email addresses [35, 52]. With the associations, Xiong et al. [52] observed that active issue committers on GitHub are also active question askers. However, Badashian et al. [35] showed that the relation between activities on the two platforms is not

strong enough, to predict developers' activities on one platform based on their activities on the other. Some studies were performed to identify the reuse of SO answers by GitHub projects, via clone detection or keyword-based search [36, 41, 47, 51, 53].

Our research also used clone detection to find answer reuse. However, different from the studies mentioned above, we did not blindly trust the clone detection results for two reasons. First, duplication does not necessarily imply code reuse. For instance, an old Java file can share code with a newly posted SO answer, although it is impossible for the file developers to take a time travel and refer to that answer posted in the future. Second, existing clone detectors can only find fragments that are very similar to each other. When code fragments are less similar, existing tools can fail to identify the reuse scenarios. To mitigate these issues of clone detection, we adopted two methods in our research. First, we manually inspected results of clone detection to remove false positives, which show code overlaps although developers actually did not reuse the answer code. Second, we also did keyword-based search to find Java files that explicitly reference the SO answers under analysis, to identify some reuse scenarios missed by clone detection.

Manes and Baysal [41, 42] mined SOTorrent and GHTorrent to locate files referencing SO posts. They looked at 30 most popularly cited SO posts with non-programming language tags, and identified the top popular tags being related to Linux OS, string, and regular expressions [41]. They also analyzed the evolution patterns of reused code snippets on SO and GitHub [42] However, neither study explores the similarity between referenced code on SO and the revised code on GitHub. Wu et al. [51] searched GitHub for source files with keyword "stackoverflow", and manually inspected retrieved files written in Java, JavaScript, Objective C, PHP, or Python. They found that in 31.5% of the data, developers modified source code from SO. In another 35.5% of cases, developers used SO posts as an information source for later reference, instead of copying any code from those posts.

As with Wu et al., we also observed different types of reuse practices by developers. However, the post distribution among reuse categories is different. Namely, we found that developers modified code from SO in a much higher percentage of scenarios (i.e., 85%); they referred to SO as an information source without copying any code in only 6% of cases. This may be because our taxonomy of reuse patterns is more comprehensive and we formulated our dataset differently. Specifically, the taxonomy of Wu et al. only includes 5 high-level categories: C1–C5; our taxonomy also includes 11 subcategories under C3, to further differentiate between developers' copy-paste-revise practices. In particular, among the 11 subcategories, we found C3.7 to be the largest one, for which developers copied and pasted most of the answer code while applying both statement-level updates and structure-level changes. Additionally, our dataset is larger, covering more unique files (i.e., 407 vs. 289) and repositories (i.e., 357 vs. 182); our dataset focused on Java code while Wu et al. studied code in five languages.

Different from all prior work, our research characterizes the reused answers from new angles like question styles, post scores, post ages, code lengths, and text lengths; our paper introduces a new and carefully designed approach to precisely locate SO answers reused by Java files on GitHub.

6 THREATS TO VALIDITY

Threats to External Validity. Our study is based on the 30,000 SO answer posts retrieved by Google BigQuery, and the Java files from GitHub citing those answers or discussion threads. Our findings may not generalize well to the threads or Java files not included in our dataset. To further investigate this threat, we did keyword-based search on GitHub, and found around 108 thousand Java files citing SO threads as of 2024. Our dataset of 407 unique Java files citing SO threads is larger than 385, the minimum sample size required to derive representative conclusions with a confidence level of 95% for the large set of SO-citing Java files. Thus, our major findings are still representative and can generalize well.

Threats to Construct Validity. We detected answer reuses in two complementary ways: clone detection and keyword-based search. There can be reuse scenarios not captured by either way, such as those having no clones and citing no answers explicitly. The actual reuse scenarios existing in our dataset may be more than what we found. We share this limitation with existing studies. As more developers explicitly reference the SO answers they reuse in codebases, such limitations can get alleviated.

Threats to Internal Validity. Our research involves manual inspection for (1) the refinement of clone-detection results and keyword-based search results, (2) topic identification for SO discussion threads, and (3) classification of developers' answer-reuse practices. Our manual analysis may be subject to human bias. To mitigate that issue, we had two authors (1) check the datasets simultaneously, and (2) discuss frequently to resolve any divergent opinions or labels.

7 CONCLUSION

We did an empirical study to characterize reused SO answers, and explore how Java developers reuse SO answers in GitHub projects. Compared with prior work, our hybrid approach is unique in two aspects. First, after using regular expressions to locate Java files that cite any SO post, it combines two methods—clone detection and keyword-based search—to detect candidate SO answers reused by Java files. As the two methods have separate strengths and weaknesses, combining them enables us to identify a high-quality set of answers potentially reused. Second, we did manual inspection to refine the candidates retrieved by both methods. With such a careful approach design, we did our study with high rigor. In the future, we plan to create a tool to crawl for related code examples on GitHub given an SO answer.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their valuable feedback. We also thank Kulendra Kumar Kaushal and Rutwik Kulkarni, for their initial contribution to the project. This work was partially supported by NSF grant NSF-1845446.

REFERENCES

[1] 2008. Calculate distance between two latitude-longitude points? (Haversine formula). https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula.
[2] 2008. Difference between StringBuilder and StringBuffer. https://stackoverflow.com/questions/355089/difference-between-stringbuilder-and-stringbuffer.
[3] 2008. How can I play sound in Java? https://stackoverflow.com/questions/26305/how-can-i-play-sound-in-java/26318#26318.

[4] 2008. How do I check CPU and Memory Usage in Java? https://stackoverflow.com/questions/74674/how-do-i-check-cpu-and-memory-usage-in-java.
[5] 2008. How do I make a list with checkboxes in Java Swing? https://stackoverflow.com/questions/19766/how-do-i-make-a-list-with-checkboxes-in-java-swing.
[6] 2008. How do I use 3DES encryption/decryption in Java? https://stackoverflow.com/questions/20227/how-do-i-use-3des-encryption-decryption-in-java.
[7] 2008. How to do query auto-completion/suggestions in Lucene? https://stackoverflow.com/questions/120180/how-to-do-query-auto-completion-suggestions-in-lucene.
[8] 2008. How to emulate C# as-operator in Java. https://stackoverflow.com/questions/148828/how-to-emulate-c-sharp-as-operator-in-java.
[9] 2008. How to validate an XML file against an XSD file? https://stackoverflow.com/questions/15732/how-to-validate-an-xml-file-against-an-xsd-file.
[10] 2008. How would you implement an LRU cache in Java? https://stackoverflow.com/questions/221525/how-would-you-implement-an-lru-cache-in-java.
[11] 2008. Java: How to test methods that call System.exit()? https://stackoverflow.com/questions/309396/java-how-to-test-methods-that-call-system-exit.
[12] 2008. Regular expression to match URLs in Java. https://stackoverflow.com/questions/163360/regular-expression-to-match-urls-in-java/163539#163539.
[13] 2008. The most efficient way to implement an integer based power function pow(int, int). https://stackoverflow.com/questions/101439/the-most-efficient-way-to-implement-an-integer-based-power-function\protect\discretionary{\char\hyphenchar\font}{}{}powint-int.
[14] 2008. Unique (non-repeating) random numbers in O(1)? https://stackoverflow.com/questions/196017/unique-non-repeating-random-numbers-in-o1#196065.
[15] 2008. What issues should be considered when overriding equals and hashCode in Java? https://stackoverflow.com/questions/27581/what-issues-should-be-considered-when-overriding-equals-and-hashcode\protect\discretionary{\char\hyphenchar\font}{}{}in-java.
[16] 2008. What issues should be considered when overriding equals and hashCode in Java? https://stackoverflow.com/questions/27581/what-issues-should-be-considered-when-overriding-equals-and-hashcode\protect\discretionary{\char\hyphenchar\font}{}{}in-java/27609#27609.
[17] 2009. Drop all tables command. https://stackoverflow.com/questions/525512/drop-all-tables-command.
[18] 2009. How do I make a list with checkboxes in Java Swing? https://stackoverflow.com/questions/19766/how-do-i-make-a-list-with-checkboxes-in-java-swing/1966151#1966151.
[19] 2009. Number of lines in a file in Java. https://stackoverflow.com/questions/453018/number-of-lines-in-a-file-in-java.
[20] 2009. Strange OutOfMemory issue while loading an image to a Bitmap object. https://stackoverflow.com/questions/477572/strange-outofmemory-issue-while-loading-an-image-to-a-bitmap-object.
[21] 2012. hive-mrc/Autocomplete.java. https://github.com/datadryad/hive-mrc/blob/master/hive-core/src/edu/unc/ils/mrc/hive/ir/lucene/search/Autocomplete.java.
[22] 2013. Amber-IDE/CheckBoxList.java at master. https://github.com/IvyBits/Amber-IDE/blob/master/src/main/java/tk/amberide/ide/swing/list/CheckBoxList.java.
[23] 2014. IoT-Gateway/LinkFormatReader.java at master. https://github.com/Ericsson-LMF/IoT-Gateway/blob/master/osgi/basedriver.coap/src/main/java/com/ericsson/deviceaccess/coap/basedriver/util/LinkFormatReader.java.
[24] 2016. ieg-util/MoreMath.java. https://github.com/ieg-vienna/ieg-util/blob/master/src/ieg/util/lang/MoreMath.java.
[25] 2016. Proyecto-Prog3/Sound.java at master. https://github.com/Tzongo/Proyecto-Prog3/blob/master/ProgIII-Proyecto/src/sounds/Sound.java.
[26] 2016. versions/Sharer.java at master. https://github.com/p2p-sync/versions/blob/master/src/main/java/org/rmatil/sync/version/core/model/Sharer.java.
[27] 2019. Karp-Rabin. https://xlinux.nist.gov/dads/HTML/karpRabin.html.
[28] 2022. 2020 Developer Survey. https://insights.stackoverflow.com/survey/2020.
[29] 2022. Files for stackexchange. https://archive.org/download/stackexchange.
[30] 2022. Hello World - GitHub Docs. https://docs.github.com/en/get-started/quickstart/hello-world.
[31] 2022. PMD. https://www.ing.iac.es//~docs/external/java/pmd/cpd.html.
[32] 2022. What is BigQuery? https://cloud.google.com/bigquery/docs/introduction.
[33] 2008. [n. d.]. How can I play sound in Java? https://stackoverflow.com/questions/26305/how-can-i-play-sound-in-java.
[34] L. An, O. Mlouki, F. Khomh, and G. Antoniol. 2017. Stack Overflow: A code laundering platform?. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Los Alamitos, CA, USA, 283–293. https://doi.org/10.1109/SANER.2017.7884629
[35] Ali Sajedi Badashian, Afsaneh Esteki, Ameneh Gholipour, Abram Hindle, and Eleni Stroulia. 2014. Involvement, Contribution and Influence in GitHub and Stack Overflow. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering* (Markham, Ontario, Canada) *(CASCON '14)*. IBM Corp., USA, 19–33.
[36] Sebastian Baltes, Richard Kiefer, and Stephan Diehl. 2017. Attribution Required: Stack Overflow Code Snippets in GitHub Projects. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 161–163. https://doi.org/10.1109/ICSE-C.2017.99
[37] M. Chen, F. Fischer, N. Meng, X. Wang, and J. Grossklags. 2019. How Reliable is the Crowdsourced Knowledge of Security Implementation?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 536–547. https://doi.org/10.1109/ICSE.2019.00065
[38] Neelamadhav Gantayat, Pankaj Dhoolia, Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2015. The Synergy between Voting and Acceptance of Answers on StackOverflow - Or the Lack Thereof. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 406–409. https://doi.org/10.1109/MSR.2015.50
[39] Jerry L. Hintze and Ray D. Nelson. 1998. Violin Plots: A Box Plot-Density Trace Synergism. *The American Statistician* 52, 2 (1998), 181–184. https://doi.org/10.1080/00031305.1998.10480559 arXiv:https://www.tandfonline.com/doi/pdf/10.1080/00031305.1998.10480559
[40] Verena Honsel, Steffen Herbold, and Jens Grabowski. 2015. Intuition vs. Truth: Evaluation of Common Myths about StackOverflow Posts. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 438–441. https://doi.org/10.1109/MSR.2015.58
[41] Saraj Singh Manes and Olga Baysal. 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)*. 235–239. https://doi.org/10.1109/MSR.2019.00047
[42] Saraj Singh Manes and Olga Baysal. 2021. Studying the Change Histories of Stack Overflow and GitHub Snippets. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 283–294. https://doi.org/10.1109/MSR52588.2021.00040
[43] Patrick E. McKnight and Julius Najab. 2010. *Mann-Whitney U Test*. John Wiley & Sons, Ltd, 1–1. https://doi.org/10.1002/9780470479216.corpsy0524 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470479216.corpsy0524
[44] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango-Argoty. 2018. Secure coding practices in java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 372–383.
[45] Dana Movshovitz-Attias, Yair Movshovitz-Attias, Peter Steenkiste, and Christos Faloutsos. 2013. Analysis of the reputation system and user contributions on a question answering website: StackOverflow. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*. 886–893. https://doi.org/10.1145/2492517.2500242
[46] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q amp;A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 25–34. https://doi.org/10.1109/ICSM.2012.6405249
[47] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. 2021. Toxic Code Snippets on Stack Overflow. *IEEE Transactions on Software Engineering* 47, 3 (2021), 560–581. https://doi.org/10.1109/TSE.2019.2900307
[48] J. T. Roscoe. 1975. *Fundamental Research Statistics for the behavioral sciences* (2. ed ed.). Holt, Rinehart & Winston, New York. http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+226871045&sourceid=fbw_bibsonomy
[49] C.B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572. https://doi.org/10.1109/32.799955
[50] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. StackOverflow and GitHub: Associations Between Software Development and Crowdsourced Knowledge. *Proceedings - SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013*, 188–195. https://doi.org/10.1109/SocialCom.2013.35
[51] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. 2019. How do developers utilize source code from stack overflow? *Empirical Software Engineering* 24, 2 (2019), 637–673. https://doi.org/10.1007/s10664-018-9634-5
[52] Yunxiang Xiong, Zhangyuan Meng, Beijun Shen, and Wei Yin. 2017. Developer Identity Linkage and Behavior Mining Across GitHub and StackOverflow. *International Journal of Software Engineering and Knowledge Engineering* 27, 09n10 (2017), 1409–1425. https://doi.org/10.1142/S0218194017400034 arXiv:https://doi.org/10.1142/S0218194017400034
[53] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. 2017. Stack Overflow in Github: Any Snippets There?. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 280–290. https://doi.org/10.1109/MSR.2017.13
[54] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are Code Examples on an Online Q&A Forum Reliable? A Study of API Misuse on Stack Overflow. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 886–896. https://doi.org/10.1145/3180155.3180260