Recommending Code Improvements Based on Stack Overflow Answer Edits*

Chaiyong Ragkhitwetsagul†
chaiyong.rag@mahidol.edu
Facutly of ICT, Mahidol University
Salaya, Nakhon Pathom, Thailand

Matheus Paixao
matheus.paixao@uece.br
State University of Ceara (UECE)
Fortaleza, Ceará, Brazil

arXiv:2204.06773v1 [cs.SE] 14 Apr 2022

ABSTRACT

Background: Sub-optimal code is prevalent in software systems. Developers may write low-quality code due to many reasons, such as lack of technical knowledge, lack of experience, time pressure, management decisions, and even unhappiness. Once sub-optimal code is unknowingly (or knowingly) integrated into the codebase of software systems, its accumulation may lead to large maintenance costs and technical debt. Stack Overflow is a popular website for programmers to ask questions and share their code snippets. The crowdsourced and collaborative nature of Stack Overflow has created a large source of programming knowledge that can be leveraged to assist developers in their day-to-day activities.
Objective: In this paper, we present an exploratory study to evaluate the usefulness of recommending code improvements based on Stack Overflow answers' edits.
Method: We propose Matcha, a code recommendation tool that leverages Stack Overflow code snippets with version history and code clone search techniques to identify sub-optimal code in software projects and suggest their optimised version. By using SOTorrent and GitHub datasets, we will quali-quantitatively investigate the usefulness of recommendations given by Matcha to developers using manual categorisation of the recommendations and acceptance of pull-requests to open-source projects.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; **Software evolution**.

KEYWORDS

Stack Overflow, Code Recommendation, Code Similarity

ACM Reference Format:
Chaiyong Ragkhitwetsagul and Matheus Paixao. 2022. Recommending Code Improvements Based on Stack Overflow Answer Edits. In *MSR '22: The 2022 Mining Software Repositories Conference, May 23–24, 2022, Pittsburgh, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

*The study was accepted at the MSR 2022 Registered Reports Track.
†Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MSR '22, May 23–24, 2022, Pittsburgh, PA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Sub-optimal code is prevalent in software systems, where developers may write low-quality code due to many reasons, such as lack of technical knowledge [27], lack of experience [7], time pressure [13], management decisions [8, 14], and even unhappiness [10]. Regardless of the reason behind sub-optimal code, it can unknowingly (or knowingly [4]) be integrated into the codebase of software systems. The accumulation of sub-optimal code without proper remediation or prevention may lead to large maintenance costs [28] and technical debt [31].

Stack Overflow[1] is a popular website for programmers to ask questions and share their code snippets. The collaborative nature of Stack Overflow has created a large source of programming knowledge. As a result, programmers commonly reuse code from answers in Stack Overflow in their software projects [20, 33, 35]. Although code snippets from Stack Overflow may contain issues such as security vulnerabilities [1], API misuses [36], and license-violating code [20, 35], the crowdsourcing nature of the website allows for constant community updates that not only optimise the answers' code but also fix potential issues [6, 30].

Several tools have been created that use the knowledge on Stack Overflow to assist developers. The tasks for which the tools were created are varied, such as providing working code examples [11], showing relevant Stack Overflow posts according to the code context in the IDE [18, 19], or improving API documentation [32]. Following this track, in this paper, we present an exploratory study to evaluate the usefulness of recommending code improvements based on Stack Overflow answers' edits. However, before we detail our proposal, we present an example from a Stack Overflow post.

1.1 Motivating Example

As a motivating example for this exploratory study, we take a close look at the Stack Overflow question (and its answers) number 40665315[2]. In the question, the developer is using the Rest Assured[3] library to assist in the writing of unit tests for a REST application developed with the Spring Boot[4] framework. For this particular unit test, the developer needs to set the port the service will be running from, which is the main topic of the question.

For this question, the same developer who asked the question figured out the solution and posted an accepted answer. The code snippet for the answer is displayed in Listing 1. The solution found by the developer can be seen in Line 10, where a method `.port()` is used to set the service's running port and test its status code.

[1]https://stackoverflow.com/
[2]https://stackoverflow.com/questions/40665315
[3]https://rest-assured.io/
[4]https://spring.io/projects/spring-boot

This solution was posted in November 2016, and this code was displayed on Stack Overflow as the accepted answer for question 40665315. After some time, another answer was posted for this question by a different user. In the alternative answer, the answerer argues that the unit test in the accepted answer was setting up the port every time the test was executed, which was a waste of resources because, if the class had multiple tests, each test would need to set the port again. To solve this, the answerer proposed the creation of a `setUp()` method that would set the port only once for all the tests in the class. Hence, in December 2018, the accepted answer was edited according to the suggestion, and its current code is displayed in Listing 2. As one can see from Lines 8 to 11, the new `setUp()` method was added, and the call for method `.port()` (Line 15) in the test was removed.

From this example, one can see that the initial solution provided as the accepted answer, albeit fully functioning, was sub-optimal. Despite the snippet in today's answer being an optimised version of the solution, the sub-optimal solution was displayed as the accepted answer for two years. During this time, all Stack Overflow users who viewed this particular question would use the sub-optimal solution as the inspiration for their solutions. Moreover, the optimisation needed to prevent the resource waste is so subtle that it would not be surprising that other developers would achieve a similar sub-optimal solution and never realise its potential problems.

For a quick assessment of this assumption, we took the statement causing the sub-optimal behaviour in the first version of the answer (`given().port(port)`), and searched GitHub's search service[5] with the statement as a string and filtering for Java projects. At the time of writing this paper, this query returned 435 results in GitHub's search. By looking at the 10 first results, we observed that 5 of them had no set up method and the port had to be set for each and every test being executed. The average number of tests in these classes was 16. In 2 out of the 10 results, there was a set up method but the port set up was not included as one of the set up steps. Finally, in 3 out of the 10 results, there was a set up method with a proper port set up, as suggested in the optimised solution.

As one can see from this example, sub-optimal solutions are posted (and accepted) on Stack Overflow, and it may take a while until the solution is optimised by the community. In the meantime, developers may use the sub-optimal snippets in their own solutions. In some cases, such sub-optimal solutions may be reached by developers on their own and not necessarily be copied from Stack Overflow. Regardless of the reason for the sub-optimal code snippets, the large crowdsourced programming knowledge on Stack Overflow can be leveraged to benefit developers.

1.2 Study's Proposal and Research Questions

In this paper, we propose an exploratory study of our approach, called Matcha, to recommend improvements to sub-optimal code based on answer edits on Stack Overflow. According to the motivating example, we can see that code snippets in a Stack Overflow accepted answer evolve over time and some of the accepted answers are later updated to include code improvements. Matcha operates in two main steps: 1) Matcha searches for similar code between a software project's code and snippets in Stack Overflow answers, 2) Matcha recommends the latest version of the answer to developers. The approach leverages the scalable code clone search technique of Siamese [20] to retrieve similar code snippets from a large corpus of Stack Overflow accepted answers. We aim to augment Siamese to handle multiple revisions of the same code snippet, which is the case in Stack Overflow answers. Importantly, we will include the recommendation module in Siamese to be able to return the latest revision of the code snippet as the search result. We foresee that this approach will be useful for developers who are unknowingly using sub-optimal snippets in their software. They can adopt the recommendation to improve the quality of their implementation. Lastly, we designed this exploratory study to evaluate the usefulness of the recommendations given by our approach.

In the study, we intend to answer the following research questions:

- **RQ1:** *How accurate is Matcha's detection of similar code snippets between software projects and Stack Overflow answers?* **(Sanity Check)**
  Objective: To check Matcha's accuracy regarding the detection of similar code snippets between software projects and Stack Overflow answers. The combination of Siamese and the additional modules (see Section 3) compose the code clone search engine employed by Matcha. Hence, this serves as a sanity check for our exploratory study because it evaluates the core underlying technology behind Matcha.
  Evaluation: We will employ the established code clone ground truth between GitHub projects and Stack Overflow posts provided by the SOTorrent dataset [2]. By leveraging this dataset, we will extract the 69,885 Java method clone pairs between GitHub and Stack Overflow to evaluate the accuracy of Matcha. We will search for the optimised configurations of Siamese and also compare the accuracy before and after integrating the *boiler-plate code filter* module required by Matcha (explained in detail in Section 3).
- **RQ2:** *To what extent are the recommendations given by Matcha useful for developers?*
  Objective: To assess how often Matcha provides recommendations with relevant code updates that are useful for developers.
  Evaluation: We will consider our own dataset of GitHub projects selected for this exploratory study (see Section 4.3). For each project in the dataset, we will run Matcha and store its recommendations. Next, the recommendations will be manually classified according to Baltes et. al's [3] categorisation of Stack Overflow post edits. Finally, the recommendations that are classified as relevant code updates (e.g., `Optimising` and `Refactoring`) will be considered useful for developers.
- **RQ3:** *To what extent are Matcha's recommendations accepted in practice?*
  Objective: To assess the potential of Matcha being adopted by real-world developers and integrated into their development practices.
  Evaluation: We will select a subset of the recommendations considered useful in RQ2. For each of the selected recommendations, we will open a pull-request to the GitHub project containing the code change and a description of the change according to the original Stack Overflow post. The outcome of the pull-requests

[5]https://github.com/search

```
1    @RunWith(SpringRunner.class)
2    @SpringBootTest(webEnvironment = SpringBootTest.
      WebEnvironment.RANDOM_PORT)
3
4    public class SizesRestControllerIT {
5        @LocalServerPort
6        int port;
7
8        @Test
9        public void test2() throws InterruptedException {
10           given().port(port).basePath("/clothes").get("
      ").then().statusCode(200);
11       }
12   }
13
14
15
16
17
```

Listing 1: Code from the original accepted answer to Stack Overflow question number 40665315

```
1    @RunWith(SpringRunner.class)
2    @SpringBootTest(webEnvironment = SpringBootTest.
      WebEnvironment.RANDOM_PORT)
3
4    public class SizesRestControllerIT {
5        @LocalServerPort
6        int port;
7
8        @Before
9        public void setUp() {
10           RestAssured.port = port;
11       }
12
13       @Test
14       public void test2() throws InterruptedException {
15           given().basePath("/clothes").get("").then().
      statusCode(200);
16       }
17   }
```

Listing 2: Code from the updated accepted answer to Stack Overflow question number 40665315

alongside the discussions between developers will be used to quali-quantitatively answer this research question.

2 BACKGROUND AND RELATED WORK

2.1 Recommending Code Snippets

In a recent study, Tang et al. [30] proposed a method to automatically identify comment-edit pairs on Stack Overflow, i.e., comments in an answer that trigger an edit in the answer. One of the usage scenarios for the comments-edit pairs envisioned by the authors is the recommendation of the edits to projects with the same outdated snippet, where the comment would be used as the natural language description of the recommendation. The approach proposed in this related work presents a few shortcomings. First, the answers' updates considered in the related work are limited to the ones triggered by comments, which excludes all other answers' edits due to other reasons, such as the one presented in our motivating example (see Section 1.1). Differently, Matcha considers all answers edits on Stack Overflow as candidates for recommendation, enlarging its pool of optimised snippets. Second, in the related work, only exact matches (Type-1 clones [24]) between snippets were considered for recommendation. By leveraging Siamese's ability to search for Type-1, Type-2 and Type-3 clones [20, 24], Matcha expands its ability to find sub-optimal snippets in software projects.

The papers by Ponzanelli et al. [18, 19] present Prompter, an IDE plugin to recommend Stack Overflow discussions based on the developer's current coding context. By leveraging existing search engines, Prompter identifies relevant discussions based on a predefined threshold. The developer working in the IDE is given the possibility to open and visualise the Stack Overflow discussion directly in the IDE. Despite not directly recommending code snippets to developers, Prompter leverages Stack Overflow's knowledge to enhance the developer experience in the IDE by providing more context to its coding decisions. On a similar note, the work by Rubei et al. [25] sets out to recommend relevant posts based on the developer's coding context. The proposed tool, called PostFinder focuses on enhancing both Stack Overflow posts and the developer's code with additional metadata to boost the matching. The results indicate how the new features enable the matching of code context to highly relevant posts. The work by Rahman et al. [23] proposes RACK, an automated tool focused on API recommendation from Stack Overflow knowledge. Different from the previously mentioned work, the RACK tool leverages natural language queries to search for relevant Stack Overflow posts.

The community working on code recommendations has been rapidly growing, with papers proposing creative ways to recommend code from sources other than Stack Overflow. Aroma [15] is a tool that offers code recommendations based on structural code search. It compares the code query's parse tree to code snippets in the index, prune the results, and cluster the remaining results to give high-quality recommendations. Keivanloo et al. [11] proposes an approach to find working code examples from a large code corpus on the Internet by using maximal frequent itemset mining and custom search ranking function. Nyamawe et al. [17] recommend refactoring solutions by using traceability and code metrics.

2.2 SOTorrent

SOTorrent [2] is the largest dataset of Stack Overflow data to date. The dataset is created from the Stack Overflow data dump augmented with the version history of Stack Overflow content. The version history can be retrieved at the level of whole post or individual post block. It also contains references of GitHub files to Stack Overflow posts. The dataset is periodically updated and the latest version is from December 2020 (version 2020-12-31) containing the content of 51,296,931 Stack Overflow posts with 81,536,422 post versions. From our initial analysis, there are 31,659 Java accepted code answers with revisions. The dataset can be accessed via Google BigQuery[6] or Zenodo[7]. Matcha uses SOTorrent as a source of code snippets to recommend code changes.

[6] https://console.cloud.google.com/bigquery?project=sotorrent-org
[7] https://zenodo.org/record/3746061

2.3 Siamese

Siamese (Scalable and Incremental Code Clone Search via Multiple Code Representations) [20] is a novel code clone search technique that is accurate and scalable to hundreds of million lines of code. The technique includes multiple code representations by transforming code into various representations to capture different types of code clones, query reduction that keeps only highly relevant terms in the code query, and a customised ranking function that allows selection of a preferred clone type to be ranked first. It offers accurate clone search with high precision and recall for Type-1 to Type-3 clones compared to other state-of-the-art code search and code clone detection tools [20]. Siamese's clone search architecture leverages the inverted index to efficiently search for code clones. It scales to a large code corpus by indexing 365M lines of code in less than a day. Each query response time is within 8 seconds. The technique is general and can be applied to several software engineering problems such as retrieving similar code snippets from a large-scale codebase, finding code clones between Stack Overflow and GitHub projects, and analysing software license violations. Currently, Siamese supports code clone search in Java language only. However, the technique can be extended to other languages by adding additional language parsers.

3 MATCHA'S DESIGN

Matcha is a code recommendation system that accepts a Java project as input and returns a list of code recommendations, i.e., snippets from Stack Overflow answers that are similar to the input snippet with the latest updates. Matcha's main component is the Siamese code clone search engine [20]. For Matcha, we will augment Siamese to include three additional modules: *boiler-plate code filter*, *multiple-code revision search*, and *latest code revision retrieval* (see Figure 1). The three modules are crucial for giving high-quality code recommendations by Matcha. Although the input is given to Matcha as a project, Matcha will read each Java file in the project, parse them, and perform the code clone search at method-level one at a time. During the clone search process of each given code query, Matcha will perform the search using Siamese by applying multiple code representation, query reduction, and clone ranking. Within the search process, we will incorporate the three additional modules to enable Matcha to return code recommendations as follows. Siamese is currently written in the Java language. Similarly, Matcha and its new modules will also be implemented in Java.

Boiler-plate Code Filter: Clones between Stack Overflow and software projects can contain a large number of boiler-plate code [22] (e.g., getters, setters, or `equals` methods) that are not useful for developers. Thus, the *boiler-plate code filter* removes boiler-plate code from the search. The filter will be created as follows. First, we will compile a list of code patterns that are considered as boiler-plate code according to the classification in the previous study [22]. For example, the `getter`, `setter`, `equals()`, `compareTo()`, `toString()` methods will be considered as boiler-plate code. Second, we will create a list of regular expressions that match with such boiler-plate code patterns. These regular expressions will be integrated into the search component of Siamese as a query filter. We will check the given code query against the list of regular expressions. If there is a match, we will skip that code snippet from performing the search, thus removing the boiler-plate code from the search results.

Multiple Code Revision Search: This module allows Matcha to search for multiple revisions of the same code snippet in a Stack Overflow answer. The *multiple code revision search* module will be created as follows. First, we will create the clone search index of Siamese by inputting all the revisions of each Stack Overflow accepted answer in Java. We will name the code snippets in each revision by concatenating the `PostID` (the unique ID of each answer), the `LocalID` (the unique ID of each code block within the code) and the `HistoryID` (the unique ID of each revision of the code block) as defined by the SOTorrent dataset. For example, the Stack Overflow answer ID 8394534 has 3 revisions[8], they will be indexed as follows. The code snippet in the original version will be saved into a file and indexed using the file name `8394534_0_0.java`. The first revision version will be created and indexed as `8394534_0_1.java`. The second revision, the latest one, will be created and indexed as `8394534_0_latest.java`. Using this naming technique, we can automatically identify from the Siamese clone search results if the matched code snippet is the latest version or not by checking from the name of the file name of the first-ranked result.

Latest Code Revision Retrieval: The *latest code revision retrieval* module allows Matcha to return the latest revision of the matched code snippet. Based on the result from the *multiple code revision search* module, Matcha will check, for each code snippet query, whether the latest version of Stack Overflow answer is returned. If not, Matcha will include the latest version of the answer into the list of code recommendations.

After finishing searching using all the methods in the given Java project, Matcha will return the list of code recommendations in a CSV format. Each record contains the file name, method name, start line, and end line of the code in the project, and the `PostID` of the Stack Overflow post that contains the latest code revision.

4 EXECUTION PLAN

Figure 1 displays the overview for our exploratory study. The study will be separated in four sequential phases (Phase 0 is not displayed in the diagram). This section describes each phase in detail.

4.1 Phase 0: Matcha's Sanity Check

As depicted in Figure 1, and fully explained in Section 4.3, Siamese and the *boiler plate code filter* compose the core technology underlying Matcha's ability to search for similar code between a software project's code and snippets in Stack Overflow answers. Hence, the first step of our study must be an assessment of how well Matcha performs in this task.

In its original study [20], Siamese's clone search accuracy has been evaluated using several error measures including mean average precision (MAP), mean reciprocal rank (MRR), precision-at-10 and recall on the OCD, SOCO, and BigCloneBench datasets [21, 29]. The evaluation results show that Siamese can search for clones with high accuracy. In Siamese's study, among other evaluations, the approach was assessed with two experiments that consider Stack Overflow and GitHub data. First, a replication study of FaCoY [12], a code-to-code search tool, is performed. However, the replication

[8] https://stackoverflow.com/posts/8394534/revisions

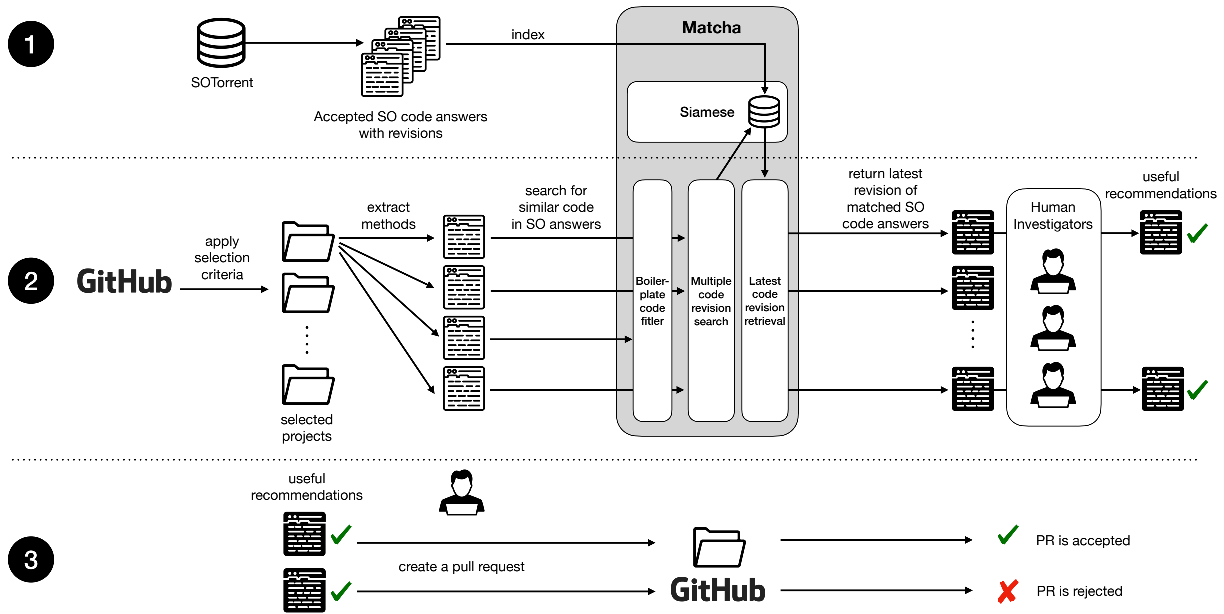

Figure 1: Overview of our exploratory study.

study only included 10 Stack Overflow code snippets as code search queries. Second, Siamese is used to analyse clones between the 72,365 Java code snippets in accepted answers on Stack Overflow and 16,738 GitHub Java projects for potential software license violations. As one can see, the Siamese clone search has not yet being directly evaluated regarding its accuracy in matching code from software projects and Stack Overflow snippets. Moreover, in a different work [22], we found that the clones between Stack Overflow and software projects may be different than normal clones, such as having incomplete code snippets and a greater number of boiler plate code. Hence, the additional *boiler plate code filter* is necessary for Matcha's design, as discussed in Section 3. Thus, it is important to evaluate Siamese in combination with the *boiler plate code filter* in a comprehensive dataset as a sanity check of Matcha's core underlying task.

For this sanity check, we will employ the established code clones ground truth between GitHub projects and Stack Overflow posts provided by the SOTorrent dataset [2]. The clone pairs are established by locating source code files in a GitHub project with a URL to a Stack Overflow post in a code comment. The `PostReferenceGH` table contains both the URL to the Stack Overflow post and the URL to the GitHub project's file. With the latest version of SOTorrent, there is a total of 6,683,852 clone pairs recorded in the table. By filtering only Java language, there is a total of 69,885 clone pairs, which we will use as our ground truth.

Next, we will extract the code snippets from both Stack Overflow answers and GitHub projects that appear in the ground truth. The code snippets in GitHub projects will be extracted at a method-level granularity, while the code snippets in Stack Overflow answers will be extracted at a file-level granularity (i.e., using the whole code snippet) because some code snippets in Stack Overflow answers are not complete methods. We will index all the Java answers in SOTorrent, not only the answers in the ground truth, in Siamese. This is to avoid the bias of searching only the true positives, which may affect the precision and recall of the results.

Moreover, since existing work [21] shows that configurations can strongly affect the performance of code clone detectors and code similarity tools, and the original configurations are not always the best one, we will also perform tuning of Siamese's configurations as follows. We will divide the clone pairs in the ground truth into two sets: the tuning set and the testing set, with the ratio of 20% and 80% of all the clone pairs accordingly. Next, we will run Siamese on the tuning set starting with the default configuration values of the following parameters: (1) code normalization, (2) n-gram size, (3) query reduction threshold. We will then perform an exhaustive search of the three parameter values to achieve the best F1 score following the method performed in our previous work [21]. We will call the configurations with the highest F1 score as the *optimised configurations.* Lastly, we will apply Siamese with the optimised configurations to the testing set and compare the results with the ground truth, create a confusion matrix, and measure the accuracy of the approach using measures such as precision, recall, and F1-score. The results will be reported in running Siamese with both original and optimised configurations.

In addition, to evaluate the effectiveness of the additional *boiler-plate code filter* module on improving the accuracy of clone search, we will also evaluate Matcha using Siamese with the optimised configurations on the testing set before and after integrating the *boiler-plate code filter* and report the comparison of the clone search accuracy (precision, recall, and F1-score) in both scenarios. Lastly, we will manually validate a sample of the clone pairs that are removed from the search results after integrating the *boiler-plate code filter* to ensure the effectiveness of the filter on removing actual boiler-plate code. Interesting findings will be reported and

discussed. The evaluation performed in Phase 0 will serve as an answer to the study's RQ1: *How accurate is Matcha's detection of similar code snippets between software projects and Stack Overflow answers?*.

4.2 Phase 1: Indexing Code Snippets from Stack Overflow's Answers on Siamese

Upon assessing Matcha's performance in matching code from software projects to snippets in Stack Overflow (Phase 0), we will move on to Phases 1-3 of our exploratory study. In Phase 1, we will prepare Matcha for the later phases. For this, we will consider the latest version of the SOTorrent dataset, as described in Section 2.2.

First, we will extract the code snippets that have at least one revision from all Java accepted answers. Next, we import the extracted snippets into the Siamese clone search index. Depending on the number of code snippets to be indexed, this step may take a while. In a previous work, the indexing of 4.8M snippets (365MLOC) in Siamese took less than a day (18 hours 13 minutes). Our initial analysis of Java posts on SOTorrent shows that there are 3,906,637 Java posts on Stack Overflow. Thus, we expect the indexing time of all the Java accepted answers to take approximately one to two days. This indexing phase occurs only once in the study and the later use of Siamese will be done by querying similar code snippets which takes a much faster time (e.g., seconds). Furthermore, by using the incremental indexing of Siamese, the expansion of the study (e.g., add more languages) will be faster compared to the initial indexing.

4.3 Phase 2: Recommending Code Updates from Stack Overflow Answer Edits

In Phase 2 of our exploratory study, we will: 1) select a set of GitHub projects as an evaluation dataset, and 2) run Matcha for each project and assess which recommendations are useful for developers. We detail each step as follows.

As previously mentioned, one of our assumptions for this study is that Stack Overflow answer edits may serve as a source for optimised code snippets with the potential to improve the codebase of software projects. In this context, it is expected that projects with different levels of code quality may be benefited differently. For instance, a project with a high-quality codebase may not have as many sub-optimal snippets, where Matcha would not find as many useful recommendations. Differently, a project with a codebase containing more sub-optimal snippets might be greatly impacted by Matcha's recommendations. To evaluate this assumption, we will need a representative sample of software projects with different levels of code quality.

To search for GitHub projects, we will employ GHS (GitHub Search) [5], a recently published tool and dataset for searching GitHub repositories that is tailored for Mining Software Repositories studies. The GHS dataset is composed of 735,669 repositories written in 10 programming languages. For each project, GHS provides data regarding 25 characteristics, which can be used as filters in the search. For this exploratory study, we will search GHS with the following filters: `Language: Java; Exclude Forks; Has Open Issues; Has Open Pull Requests`. We will only select projects written in Java due to Siamese's limitations (see Section 2.3). We will exclude all forks to make sure there will be no redundant projects in our dataset. We will select projects with both open issues and open pull-requests to guarantee the projects make use of GitHub's social features to maximise the chances to have Phase 3's pull-requests properly evaluated.

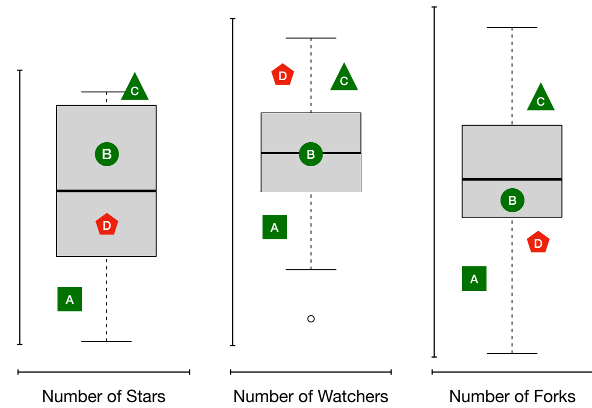

Figure 2: GitHub project selection criteria based on the distributions of number of stars, watchers, and forks.

For each project, GHS provides 3 popularity metrics: `Number of Stars`, `Number of Watchers` and `Number of Forks`. GitHub popularity metrics, such as the ones provided by GHS, have been used countless times as quality proxies for GitHub projects [9, 26, 34]. Although isolated popularity metrics have been shown not to be the most effective method to assess a project's code quality [16], we believe a combination of these metrics has the potential to yield a more trustworthy proxy. Hence, for each quality metric provided by GHS, we compute the distribution and divide it into quartiles. The projects that appear above the third quartile for all metrics will be considered as having a higher quality codebase. Similarly, the projects appearing below the first quartile for all metrics will be considered as having a lesser quality codebase. Finally, a project that appears in between the first and third quartiles for all metrics will be considered as having a medium quality codebase. Projects that appear in the intersection of quartiles between metrics, e.g., above the third quartile for a metric and below the first quartile for another metric, will be excluded from this study. These criteria will provide a clear separation of projects with different levels of code quality based on the popularity metrics proxy we will employ.

Figure 2 depicts our project selection criteria. Suppose the boxplots represent the distribution of the number of stars, number of watchers, and number of forks for all the projects that pass our GHS filter. There are four projects represented by four symbols: Project A (square), Project B (circle), Project C (triangle), and Project D (pentagon). According to our project selection criteria, we will include Project A since it appears below the first quartile for all the metrics. Similarly, we will include Projects B and C since they appear between the first and third quartiles and above the third quartile for all the metrics accordingly. We will not include Project D because it appears in a different quartile for at least two metrics.

After applying the selection criteria discussed above, we still expect to have a population of thousands of projects, which would be infeasible to consider in our study due to the manual and qualitative

methods in our evaluation. Hence, we will compute the distribution of software projects into the three code quality tiers and take a random stratified representative sample in the 95% confidence level. Due to projects' constant evolution within GitHub, we will select the projects at the time of executing this exploratory study. This way, we will have the most up to date selection of projects according to our predefined methodology.

After selecting the projects, we will run MATCHA for each project. MATCHA's execution works as follows. First, for a given software project, MATCHA will extract the code for all the projects' methods and apply the *boiler-plate code filter*. Next, for each remaining method, MATCHA will search for similar code snippets on Siamese's index, which contains the code for Stack Overflow's accepted answers with all the versions, as shown in Phase 1. If Siamese returns a snippet corresponding to an older version of a Stack Overflow answer, i.e., not the latest version, MATCHA will mark this method for recommendation. Finally, for each marked method, MATCHA will return the latest version of the answer as a recommendation.

After running MATCHA for each selected project and collecting all recommendations, we will enter the final step of Phase 2. For each recommendation, we will look at the original Stack Overflow answer and perform a manual classification according to the answer edit categorisation proposed in a related paper [3]. Each category describes the type of edit being made in the Stack Overflow answer, such as `Optimising` and `Refactoring`. Since MATCHA's recommendation is based on the latest answer edit on Stack Overflow, the category of the answer edit will also be considered the category of the recommendation provided by MATCHA. Consider the example provided in Section 1.1. By looking at the type of code changes performed between Listings 2 and 1, one would categorise the edit as `Optimising`, which would also be the category of the recommendation proposed by MATCHA. For a full reference of the categorisation, we refer to the work by Baltes et al. [3].

The manual classification will be performed independently by two researchers, where they will label each recommendation into one or more categories. If the recommendation does not fit into any of the categories previously defined, the researcher may propose a new category. After finishing their independent classification, the researchers will compare their labelling, and the inter-labeller reliability will be measured with Cohen's kappa coefficient. For the recommendations with a disagreement in classification, a third researcher will intervene to provide a final classification. After the manual classification, there will be a list of categories into which all recommendations will have been categorised. Finally, all recommendations that perform a relevant code update, such as `Optimising` and `Refactoring`, for example, will be considered useful for developers. The list of all categories considered useful for developers can only be known after the complete manual analysis is finished. Hence, it is not possible, at the time of writing this paper, to present the complete list of useful categories for recommendation.

The results obtained in Phase 2 of our exploratory study will answer RQ2: *To what extent are the recommendations given by MATCHA useful for developers?*.

4.4 Phase 3: Submitting MATCHA's Recommendations as Pull-Requests

To assess MATCHA's potential to be adopted by software developers in their projects, we need to evaluate how the recommendations are received by developers. For this, we will consider the results of RQ2 and collect the recommendations considered useful for developers. Next, we will need to select a subset of recommendations to send the pull-requests. Considering the population of all useful recommendations stratified by the projects' level of code quality and recommendations' categories (see Section 4.3), we will extract a random stratified relevant sample at the 95% confidence level.

For each recommendation in the sample, we will execute the following safeguard procedure to submit a pull-request that minimises potential issues that may be caused by our lack of knowledge regarding the project. First, we will check whether the project provides an automated test suite. If a test suite is not provided, the recommendation will be discarded from this evaluation phase. Next, after incorporating the changes according to the recommended snippet, we will run the project's entire test suite. If any test fails, the recommendation will be discarded from this evaluation phase. For the changes that pass all tests, we will submit a pull-request that incorporates the recommended changes. The description for the pull-request will be either adapted from the Stack Overflow edit comment or created by ourselves if no edit comment was provided.

Next, we will track the status of each pull-request and avoid interfering with the review process as much as possible. This is to mitigate any bias that may be included in the reviewing process in case we mention the exploratory study, the scientific methodology behind the pull-request or any other details outside the code change itself. The outcome of the pull-requests alongside the discussions between developers during review will be used to answer RQ3: *To what extent are MATCHA's recommendations accepted in practice?*.

5 THREATS TO THE VALIDITY

Internal validity: A potential threat to internal validity stems from the adoption of Siamese to search for similar code snippets in Stack Overflow accepted answers, which may contain false positives and false negatives. We will mitigate this threat in Phase 0 of our study by performing a sanity check of the code clone search accuracy of Siamese to evaluate if it is suitable for our approach. In addition, GitHub's popularity metrics (i.e. `Number of Stars`, `Number of Watchers` and `Number of Forks`) may not accurately represent a project's code quality. Nevertheless, we argue that a categorisation criteria that combines all three metrics and excludes projects that intersect metrics can yield a more trustworthy proxy. Lastly, the acceptance or rejection of pull-requests may not fully represent MATCHA's potential of acceptance by developers. We will mitigate this threat by performing a qualitative analysis of the pull-requests and the discussions between developers during review to understand the actual reason for accepting or rejecting such pull-requests.

External validity: The ground truth data in Phase 0 is based on the code snippets in GitHub projects that have a comment pointing to the original Stack Overflow post where the code is copied from. The clones can potentially be biased to only clones that are copied from Stack Overflow with attributions and may not be generalised

to all the clones between Stack Overflow and GitHub. However, to the best of our knowledge, it is the only ground truth that contains a large amount of clone pairs (6,9885) with real evidence that the code has been reused from Stack Overflow to GitHub. Moreover, by relying on the code comments created by the authors of the code, we avoid the threat of manually labelling the clones by ourselves. The conclusions drawn from this study will be based on the selection of GitHub projects according to our predefined methodology detailed in Phase 2. It may not generalise to all GitHub projects. Moreover, we will only analyse code written in Java in both GitHub and Stack Overflow answers. Hence, the findings may not be applied to other programming languages. Finally, in this study, we will focus on accepted answers, where the findings may not be applicable to other types of Stack Overflow answers (e.g., newest answers, highest-voted answers).

6 CONCLUSION

This paper presents an exploratory study of recommending code improvements for sub-optimal code snippets based on the latest edit of Stack Overflow accepted answers. We propose an approach called Matcha that can search for similar code snippets in several revisions of Stack Overflow accepted answers and recommends code changes to improve the quality of a software project's code snippets. We plan to evaluate our approach by performing manual validation on the usefulness of the code recommendations provided by Matcha and by submitting pull-requests containing the useful recommendations to the GitHub projects. This paper also presents the methodology for selecting GitHub projects that we intend to use in our planned study. We expect the results from the exploratory study will shed light on the potential of leveraging Stack Overflow answers for code recommendation.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *IEEE Symposium on Security and Privacy (SP '16)*. 289–305.

[2] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. 2018. SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. 319–330.

[3] Sebastian Baltes and Markus Wagner. 2020. An annotated dataset of stack overflow post edits. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20)*, Vol. 323. ACM, 1923–1925.

[4] Gabriele Bavota and Barbara Russo. 2016. A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. 315–326.

[5] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *Proceedings of the 18th International Conference on Mining Software Repositories (MSR '21)*. 560–564.

[6] Themistoklis Diamantopoulos, Maria Ioanna Sifaki, and Andreas Symeonidis. 2019. Towards Mining Answer Edits to Extract Evolution Patterns in Stack Overflow. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR '19)*, Vol. 2019-May. 216–219.

[7] Jon Eyolfson, Lin Tan, and Patrick Lam. 2011. Do time of day and developer experience affect commit bugginess. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*. 153.

[8] Sávio Freire, Nicolli Rios, Boris Gutierrez, Darío Torres, Manoel Mendonça, Clemente Izurieta, Carolyn Seaman, and Rodrigo O. Spínola. 2020. Surveying Software Practitioners on Technical Debt Payment Practices and Reasons for not Paying off Debt Items. In *Proceedings of the Evaluation and Assessment in Software Engineering (EASE '20)*. 210–219.

[9] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. The State of the ML-universe. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20)*. ACM, 431–442.

[10] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2017. Unhappy Developers: Bad for Themselves, Bad for Process, and Bad for Software Product. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17)*. 362–364.

[11] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. 664–675.

[12] Kisub Kim, Dongsun Kim, Tegawendé F Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCoY – A Code-to-Code Search Engine. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. 946–957.

[13] Miikka Kuutila, Mika Mäntylä, Umar Farooq, and Maëlick Claes. 2020. Time pressure in software engineering: A systematic review. *Information and Software Technology* 121 (May 2020), 106257.

[14] Mathieu Lavallee and Pierre N. Robillard. 2015. Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality. In *Proceedings of the 37th International Conference on Software Engineering (ICSE '15)*, Vol. 1. 677–687.

[15] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. 2019. Aroma: Code Recommendation via Structural Code Search. *Proceedings of the ACM on Programming Languages* 3, OOPSLA, Article 152 (2019), 28 pages.

[16] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empirical Software Engineering* 22, 6 (Dec 2017), 3219–3253.

[17] Ally S. Nyamawe, Hui Liu, Zhendong Niu, Wentao Wang, and Nan Niu. 2018. Recommending Refactoring Solutions Based on Traceability and Code Metrics. *IEEE Access* 6 (2018), 49460–49475.

[18] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack Overflow in the IDE. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. 1295–1298.

[19] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR '14)*. 102–111.

[20] Chaiyong Ragkhitwetsagul and Jens Krinke. 2019. Siamese: scalable and incremental code clone search via multiple code representations. *Empirical Software Engineering* 24 (8 2019), 2236–2284. Issue 4.

[21] Chaiyong Ragkhitwetsagul, Jens Krinke, and David Clark. 2018. A comparison of code similarity analysers. *Empirical Software Engineering* 23, 4 (August 2018), 2464–2519.

[22] 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.

[23] Mohammad Masudur Rahman, Chanchal K. Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER '16)*. 349–359.

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

[25] Riccardo Rubei, Claudio Di Sipio, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow posts to support software developers. *Information and Software Technology* 127 (Nov 2020), 106367.

[26] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. 2014. Understanding "watchers" on GitHub. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR '14)*. 336–339.

[27] Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. 2020. Here We Go Again: Why is It Difficult for Developers to Learn Another Programming Language?. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. 691–701.

[28] Dag I.K. Sjoberg, Aiko Yamashita, Bente C.D. Anda, Audris Mockus, and Tore Dyba. 2013. Quantifying the Effect of Code Smells on Maintenance Effort. *IEEE Transactions on Software Engineering* 39, 8 (aug 2013), 1144–1156.

[29] Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME '14)*. 476–480.

[30] Henry Tang and Sarah Nadi. 2021. On using Stack Overflow comment-edit pairs to recommend code maintenance changes. *Empirical Software Engineering* 26, 4 (Jul 2021), 68.

[31] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86, 6 (Jun 2013), 1498–1516.

[32] Christoph Treude and Martin P Robillard. 2016. Augmenting API documentation with insights from stack overflow. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM Press, 392–403.

[33] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. 2017. Stack Overflow in GitHub: Any Snippets There?. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR '17)*.

[34] Fiorella Zampetti, Gabriele Bavota, Gerardo Canfora, and Massimiliano Di Penta. 2019. A Study on the Interplay between Pull Request Review and Continuous Integration Builds. In *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER '19)*. 38–48.

[35] Haoxiang Zhang, Shaowei Wang, Tse hsun Peter Chen, Ying Zou, and Ahmed E Hassan. 2019. An Empirical Study of Obsolete Answers on Stack Overflow. *IEEE Transactions on Software Engineering* (2019), 850–862.

[36] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are code examples on an online Q&A forum reliable?. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. 886–896.