Can Issues Reported at Stack Overflow Questions be Reproduced? An Exploratory Study

Saikat Mondal, Mohammad Masudur Rahman, Chanchal K. Roy
Department of Computer Science, University of Saskatchewan, Canada
{saikat.mondal, masud.rahman, chanchal.roy}@usask.ca

***Abstract*—Software developers often look for solutions to their code level problems at Stack Overflow. Hence, they frequently submit their questions with sample code segments and issue descriptions. Unfortunately, it is not always possible to reproduce their reported issues from such code segments. This phenomenon might prevent their questions from getting prompt and appropriate solutions. In this paper, we report an exploratory study on the reproducibility of the issues discussed in 400 questions of Stack Overflow. In particular, we parse, compile, execute and even carefully examine the code segments from these questions, spent a total of 200 man hours, and then attempt to reproduce their programming issues. The outcomes of our study are two-fold. First, we find that 68% of the code segments require minor and major modifications in order to reproduce the issues reported by the developers. On the contrary, 22% code segments completely fail to reproduce the issues. We also carefully investigate why these issues could not be reproduced and then provide evidence-based guidelines for writing effective code examples for Stack Overflow questions. Second, we investigate the correlation between issue reproducibility status (of questions) and corresponding answer meta-data such as the presence of an accepted answer. According to our analysis, a question with reproducible issues has at least three times higher chance of receiving an accepted answer than the question with irreproducible issues.**

***Index Terms*—Issue reproducibility, Stack Overflow, code segments, code level modifications, reproducibility challenges**

I. Introduction

Stack Overflow has become the ultimate platform for developers [15] and the number of users and the questions posted are increasing exponentially [2]. A large number of questions contain such code segments that could potentially have issues (e.g., error, unexpected behaviour) [25]. Once submitted, users at Stack Overflow generally attempt to *reproduce* the programming issues discussed in the questions, and then submit their solutions. Reproducibility means a complete agreement between the reported issues and the investigated issues [28]. Unfortunately, such programming issues could always not be reproduced [14, 21] by other users which might prevent these questions from getting appropriate and prompt answers. Such scenario might also explain the 13.36% unanswered and 47.04% unresolved questions at Stack Overflow [2, 3, 17]. Given these major challenges in question answering, a detailed investigation is warranted on the *reproducibility* of the code level issues that are reported at Stack Overflow questions. Several existing studies [11, 29] investigate the usability and executability of the code segments posted on Q&A sites. Yang et al. [29] extract 914,974 code segments from the accepted answers of Stack Overflow and analyse their parsability and compilability. However, their analysis was completely automatic, and they did not address the challenges of issue reproducibility. Horton and Parnin [11] analyse the executability of the Python code segments found on GitHub Gist system. They identify several flaws (e.g., import error, syntax errors, indentation error) that prevent the execution of such code segments. However, a simple execution success of the code does not necessarily guarantee the reproduction of the issues discussed at Stack Overflow questions. Thus, their approach also fails to address the reproducibility challenges that we are dealing with. In short, all challenges of reproducibility could not be resolved using only automated analysis. Thus, there is a marked lack of research that carefully investigates (1) the challenges of issue reproducibility (of Stack Overflow questions) and (2) the approaches to overcome such challenges.

In this paper, we report an exploratory study on the *reproducibility* of the discussed programming issues at 400 questions from Stack Overflow. For each of these questions, we follow three logical and sequential steps. First, we attempt to understand the issue of the question by analysing its code segment and the associated textual description. Second, we clone the code segment on our development environment (e.g., IDE), and attempt to reproduce the reported issue. Third, we record our findings on the reproducibility of the issues from each of the questions. In the case of failure, we investigate why the issues could not be reproduced. We spent a total of 200 man-hours for our analysis in this study. Our findings from this study are two-fold. First, we find that 68% issues reported at Stack Overflow questions can be reproduced and only 32% of them can be reproduced using the verbatim code from Stack Overflow. The remaining code segments warrant either minor or major modifications in order to reproduce the issues. Second, we investigate the relationship between the reproducibility status of the issues (from questions) and the answer meta-data (e.g., accepted answer). We found that a question with reproducible issues has more than three times higher chance of getting an acceptable answer than the question with irreproducible issues. Our in-depth investigation and findings not only establish *issue reproducibility* as a question quality paradigm but also encourage automated tool supports for improving the code segments submitted at Stack Overflow. We answer three research questions and thus make three contributions in this paper as follows:

(RQ_1) What are the challenges in reproducing the issues reported at Stack Overflow questions? How can

issue reproducibility be measured?
We conduct extensive manual analysis and investigate what types of actions are needed to reproduce the issues reported at Stack Overflow questions. We classify the reproducibility status into two major categories (*reproducible, irreproducible*) and also discuss why the reported issues could not be reproduced using the submitted code examples.

(RQ$_2$) What proportion of reported issues at Stack Overflow questions can be reproduced successfully?
We conduct a detailed statistical analysis to determine what percentage of the issues can be reproduced and what percentage cannot be reproduced even after performing major modifications to their corresponding code segments.

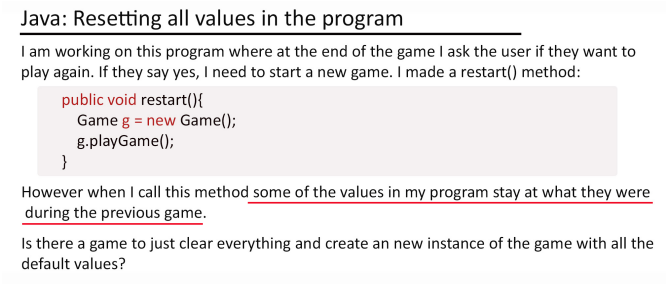
(RQ$_3$) Does reproducibility of issues reported at Stack Overflow questions help them get high-quality responses including the acceptable answers?
We conduct a detailed investigation and determine the correlation between issue reproducibility (of questions) and corresponding answer meta-data such as presence of an accepted answer, time delay between question and accepted answer, and the number of answers.

II. Motivating Examples

Code segments submitted as a part of the questions at Stack Overflow might always not be sufficient enough to reproduce the reported issues. Let us consider the example question in Fig. 1. Here, the user attempts to reset all the variables while starting a new game. He discovers that the code is not working as expected and some of the variables are retaining their old values. Unfortunately, this issue cannot be reproduced since essential parts of the code are missing. For example, one user commented – *"Can you post more code? The Game class? The class that contains the restart() method?"* – while attempting to answer the question. In Stack Overflow, this question has failed to receive a precise response. Even though the above code (i.e., Fig. 1) could be made parsable, compilable and executable with all necessary editing, the above reported issue could not be reproduced easily due to its complex nature. Thus, the automated analyses done by the earlier studies [11, 29] might also not be sufficient enough to overcome all the challenges of this reproducibility.

Let us consider another example question as shown in the Fig. 2. Suppose Alice, a software developer, wants to answer this question. First, she attempts to identify the issue and soon understands that the user is attempting to capture a person's full name (e.g., John Doe). Unfortunately, the user is getting only the first part of the given name (e.g., John). He invoked `next()` method of the `Scanner` class to take the input. In order to answer, Alice first copies the code segment to the IDE and then finds that the code does not even parse. As a result, the IDE returns several parsing and compilation errors.

[1]https://stackoverflow.com/questions/798184
[2]https://stackoverflow.com/questions/19509647


Java: Resetting all values in the program

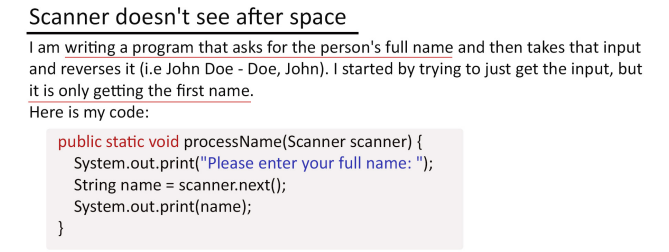
I am working on this program where at the end of the game I ask the user if they want to play again. If they say yes, I need to start a new game. I made a restart() method:

```
public void restart(){
   Game g = new Game();
   g.playGame();
}
```

However when I call this method some of the values in my program stay at what they were during the previous game.

Is there a game to just clear everything and create an new instance of the game with all the default values?

Fig. 1: An example question[1] of Stack Overflow. (**Reproducibility status:** *Irreproducible*)


Scanner doesn't see after space

I am writing a program that asks for the person's full name and then takes that input and reverses it (i.e John Doe - Doe, John). I started by trying to just get the input, but it is only getting the first name.
Here is my code:

```
public static void processName(Scanner scanner) {
   System.out.print("Please enter your full name: ");
   String name = scanner.next();
   System.out.print(name);
}
```

Fig. 2: An example question[2] of Stack Overflow. (**Reproducibility status:** *Reproducible*)

Fortunately, by performing several edits (e.g., addition of a demo class and main method) Alice could reproduce the stated issue. Similarly, this issue was also reproduced by the other users of Stack Overflow, and the question received a high-quality answer within a couple of minutes. As the solution suggests, the user should have used `nextLine()` method instead of `next()` to avoid the reported issue.

III. Study Methodology

Fig. 3 shows the schematic diagram of our conducted exploratory study. We first randomly select 400 questions of Stack Overflow, and then attempt to reproduce their discussed programming issues. In particular, we use their submitted code segments, perform a list of edits, and then attempt to make them reproduce the reported issues. The following sections discuss different steps of our methodology.

A. Dataset Preparation

Fig. 4 depicts the data collection steps. We collect November 2018 data dump of Stack Overflow from Stack Exchange site [2]. In particular, we select Java related questions since Java is one of the most popular and widely used programming languages. We collect a total of 85,876 questions that have only one tag namely <*java*> from the data dump. It should be noted that we impose this restriction on the question tags to (1) choose purely Java related questions and (2) keep our analysis simple. We then discard such questions that do not have any code segments. Since we deal with the reproducibility of the reported issues, the presence of code segments in the question is warranted. We thus consider only such questions that have at least one line of true Java code. According to our investigation, 70,103 out of 85,876 (i.e., 81.63%) questions have at least one line of code. We identify such questions

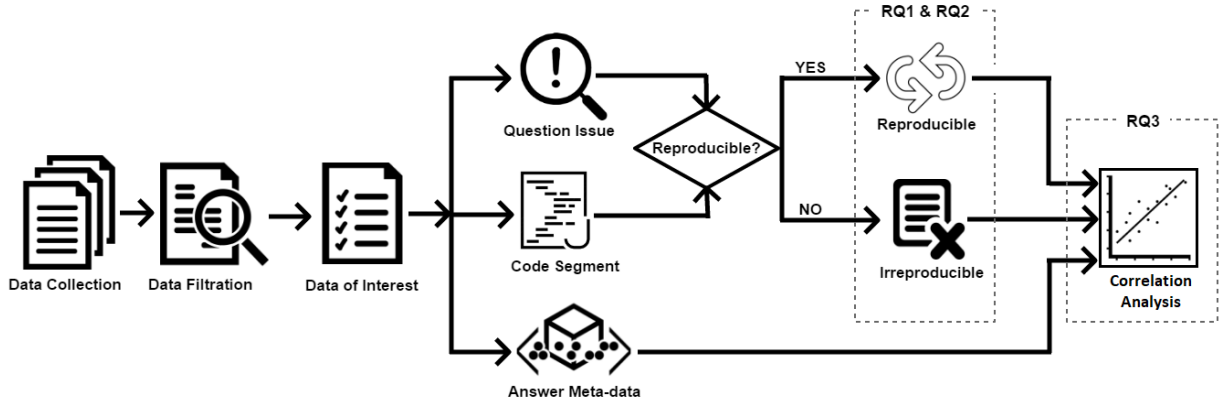


Fig. 3: Schematic diagram of our exploratory study.

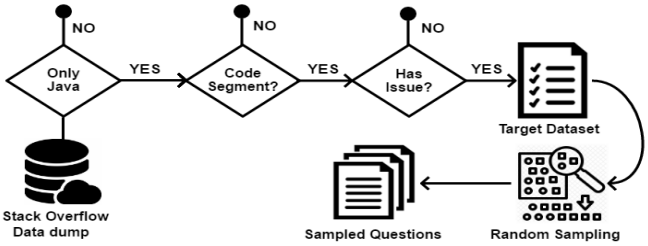


Fig. 4: Selection of dataset for our study.

TABLE I
LIST OF CODE EDITING ACTIONS

- Exception handling	- Invocation of methods
- Code migration	- Debugging
- Addition of demo classes and methods	
- Inclusion of native and external libraries	
- Object creation, identifier declaration and initialisation	
- Deletion of redundant and erroneous statements	
- External file, database and dataset creation	

using specialised HTML tags such as `<code>` under `<pre>`, and select them for our study.

Questions containing code segments might not always have an issue that needs to be reproduced during the answering time. Programmers might even seek general help or ask for more efficient source code than the submitted code segment. However, in this study, we target only such questions that discuss at least one issue each and contain at least code segment each. We thus carefully analyse the questions, and look for several keywords such as - *error, warning, issue, exception, fix, problem, wrong, fail* in the question texts. We then identify a total number of 30,528 questions that might have an issue. We randomly select 1,000 questions from them for manual analysis.

During manual analysis, we found a significant number of duplicate questions. In such cases, we consider the original questions with explicit issues and discard the duplicate questions. We also discard a few questions that were closed by Stack Overflow due to their low quality. Overall, we manually investigate 400 Java related questions from 1000 random samples that meet our selection criteria, and answer our research questions.

Replication: All experimental data and relevant materials are hosted online [1] for replication or third party reuse.

B. Environment Setup

We use `Eclipse Oxygen.3a Release (4.7.3a)`[3] and `NetBeans 8.2`[4] to execute the code segments and reproduce the programming issues. Eclipse and NetBeans are two popular IDEs that are frequently used for Java programming. When the issues are related to compilation errors, we first employ `javac` to detect the compilation problems, and then use Eclipse and NetBeans to reproduce the issues. In particular, we attempt to find an exact match between our error messages and the ones mentioned in the questions of Stack Overflow. We use `Java Development Kit(JDK)-1.8` as our development framework, `JavaParser`[5] as our custom AST parser, and MySQL Workbench[6] 8.0 as our example database for this study. We use a desktop computer having 64-bit Windows 10 Operating System (OS) and 16GB primary memory (i.e., RAM). We allocate 4GB as Java memory (Java heap for the IDE.).

C. Qualitative Analysis

We, two of the authors, took part in the manual analysis and spent a total of 200 man-hours on the 400 questions. We follow a two-step approach for reproducing the issues reported at Stack Overflow questions. First, we attempt to understand the reported issues clearly and identify the key problem statements from the question description. We also gather the supporting data such as input-output values, file format from the question texts. Second, using the code snippet and supporting data we attempt to reproduce the reported issues. We perform trivial, minor and major edits on the code segments in order to reproduce their issues. Table I shows our list of editing actions. We discuss each of the actions that are

[3]https://www.eclipse.org
[4]https://netbeans.org
[5]http://javaparser.org
[6]https://www.mysql.com/products/workbench

taken during qualitative analysis to make the code segments reproduce the reported issues as follows:

(a) *Addition of Demo Classes and Methods*: In Stack Overflow, users often submit only the code examples of interest which are neither complete nor compilable. We wrap such code segments with a demo class and then place them under a main method. Main method acts as the entry point to the program.
In some cases, the code segment contains statements related to the creation of an object of a class or method invocation using an object while the definitions of the class and the method are absent. For instance, one of our code snippets has the following statements:

```
A obj_A = new A(x,y);
obj_A.add();
```

We see that the definition of the `class A` is absent. In order to make this code compilable, we define the class, add a constructor and also define the method add() within the class. Although the actual implementation is unknown, such modifications help us to resolve the compilation errors.

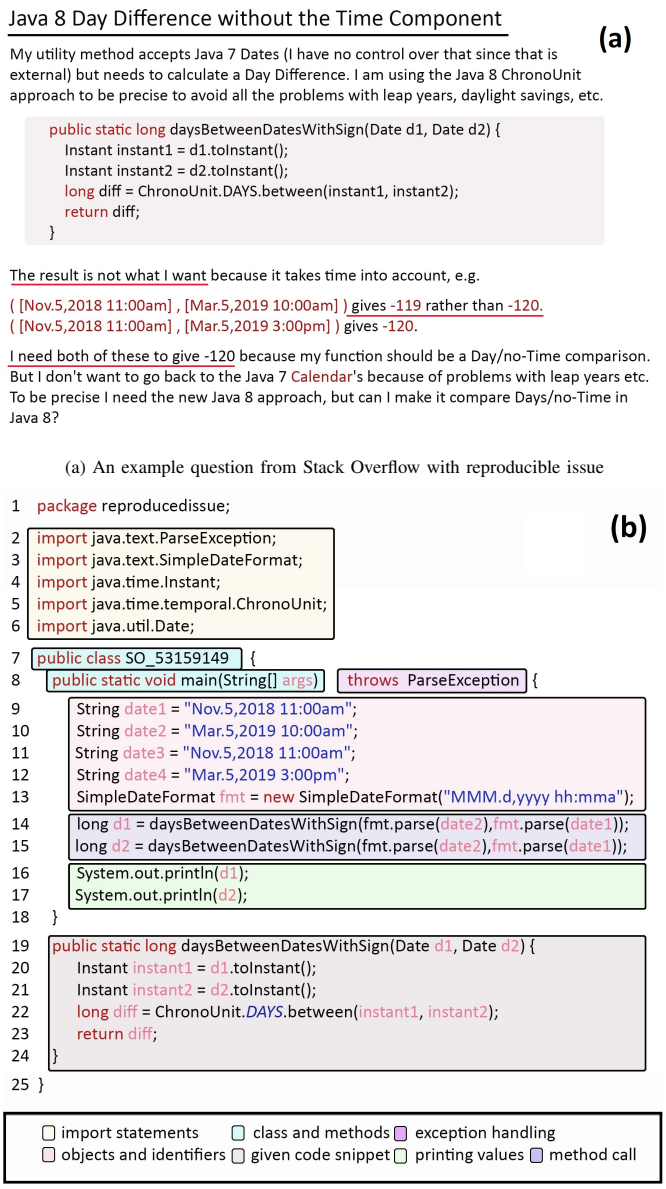
(b) *Inclusion of Native and External Libraries*: JDK comes with many libraries that help the developers accomplish many of the common tasks. However, developers also frequently use external libraries for various specialized tasks. In Stack Overflow questions, code segments that use classes and methods from native or external libraries often miss the import statements. We add the import statements associated with native libraries with the help of the IDE (e.g., Eclipse). In the case of external libraries, we look for relevant library references in the question texts. If such libraries were found, we include them in the IDE and then add necessary import statements.

(c) *Exception Handling*: Developers often submit question claiming that the code generates unexpected exception. We also find some questions whose issues are not related to Java exception but their code throws one or more exceptions. In that cases, we resolve them using appropriate exception handling.

(d) *Object Creation, Identifier Declaration and Initialization*: Undeclared identifier is one of the common compilation errors for the code segments submitted at Stack Overflow. We declare the unresolved identifiers according to their types and initialize them with appropriate values according to the question specifications.

(e) *Invocation of Methods*: We identify several code snippets that have user defined methods but the methods were not invoked from anywhere. We add extra statements to call the methods if the issue reproduction warrants the execution of these methods. We also add appropriate parameters to call the methods.

[7]https://stackoverflow.com/questions/53159149



(a) An example question from Stack Overflow with reproducible issue

(b) The final code after necessary editing

Fig. 5: An example question[7] from Stack Overflow and the final source code after editing. (**Reproducibility status:** *Reproducible*)

(f) *Deletion of Redundant and Erroneous Statements*: We find several code segments that contain redundant and even erroneous code statements which are not related to the reported issues. We delete such statements or simply comment them out so that they are ignored by compilers and interpreters.

(g) *Code Migration*: We identify several code snippets that use outdated APIs which are not compatible with our environment. We replace the outdated APIs with the equivalent updated APIs. In some cases, we also invoke extra APIs to make the code compilable.

(h) *External File, Database and Dataset Creation*: We identify several code segments that accept input from `.txt` or `.csv` file. In such cases, we create one or more necessary

files in our local drive and add the sample data from the question so that we can verify the program correctness. Besides text file or spreadsheet, some programs require image file especially which are related to User Interface (UI). In such cases, we create images with specified format and dimension, and then execute the program. We also create a demo relational database (e.g., MySQL) and necessary tables when the code segments deal with database operations. Needless to say, we use the sample dataset provided by the question.

(i) *Debugging*: We identify several code segments that warrant for debugging to reproduce the issue. Sometimes programmers claim that they are getting unexpected behaviour from the code. For example, a developer reports that one of the `if` statements is never executed i.e., the expression of the `if` is always been false. Then we debug the code and check whether the if statement is really executed. When programmers claim that they are not getting expected values from an identifier, we usually print the value or debug the code to check the run time value of the identifier.

D. An Example of Issue Reproduction

Let us consider the question shown in the Fig. 5a. First, we attempt to identify the issue and soon understand that the user is trying to calculate the time delay (in days) between two given dates. Unfortunately, the user did not get the expected results for several input values. We then copy the code segment in the IDE and find that the code does not even parse. As a result, the IDE returns several parsing and compilation errors. Fortunately, by performing several actions we can reproduce the findings. First, we add a demo class (Fig. 5b, line No. 7) and place the method `daysBetweenDatesWithSignwith` within the class. Second, we add a main method (Fig. 5b, line No. 8). Third, we create four `String` objects (Fig. 5b, line No. 9-12) and initialize them with sample values according to the question description. Fourth, we then create an object `fmt` (Fig. 5b, line No. 13) of the class `SimpleDateFormat` to invoke the parse API that converts the text values stored in `date1, date2, date3` and `date4` to the type `Date`. Since `SimpleDateFormat` throws `ParseException`, we also handle the exception (Fig. 5b, latter part of line No. 8). Afterward, we invoke the given method and keep the return values in `d1` and `d2` of type `long` (Fig. 5b, line No. 14-15). Two inline variable declarations are also required here. We import the libraries for the classes `SimpleDateFormat, Date, ParseException, Instant` and also the enum `ChronoUnit` (Fig. 5b, line No. 2-6). We then execute the modified code, print the outputs and then check whether the outputs match with the ones reported by the user. Interestingly, the issue was reproducible.

IV. Study Findings

We collect 400 questions from Stack Overflow (Section III-A), and analyse their code segments and textual descriptions. In particular, we attempt to reproduce the issues reported in these question texts by executing their corresponding code segments. We also ask three research questions in this study, and answer them carefully with the help of our empirical and qualitative findings as follows:

TABLE II
Challenges Preventing Issue Reproduction

Identified Reason	Percentage
Class/Interface/Method not found	51.00%
Important part of code missing	22.99%
External library not found	20.69%
Identifier/Object type not found	14.94%
Too short code snippet	12.64%
Miscellaneous	6.90%
Database/File/UI dependency	4.60%
Outdated code	1.15%

A. Answering RQ_1

RQ_1(a): What are the challenges in reproducing the issues reported at Stack Overflow questions? We attempt to reproduce the reported issue discussed in the question using the given code segment and other supporting information (e.g., data, instruction). However, we fail to reproduce them due to several non-trivial challenges. First, programmers often submit code segments that use methods from the classes which are not defined in the code segment. Despite adding appropriate class and method definitions, many issues cannot be reproduced. Second, code segments often miss such statements that are essential to reproduce the issues. For instance, reproducibility of an issue depends on the values of an array which are absent from the code segment. We could not reproduce such issues with the sample array values. Third, dependency on the external libraries is another major challenge towards issue reproducibility from the submitted code. In many cases, we do not find any hints that point to the appropriate libraries. We also identify such code segments that are too short to reproduce the issue. Too short code is also identified as the reason for getting no answer to Stack Overflow questions [3]. We also identify several code segments that could not reproduce the issues due to their complex interactions with UI elements, databases and external files. We find a few segments containing outdated code (e.g., deprecated API) that cannot be replaced by alternative one. Several code segments are too long and noisy to reproduce the issues.

Table II shows the major challenges that prevent the reproduction of issues reported at Stack Overflow questions. A question might experience multiple challenges. We note that half of irreproducible issues are plagued by undefined classes, interfaces and methods. That is, they are used in the code segments without proper definitions. We also note that about 40% issues could not be reproduced due to their missing import statements and missing external libraries.

RQ_1(b): How can issue reproducibility be measured? We divide the issue reproducibility status (of questions) using

[8]https://stackoverflow.com/questions/1715533

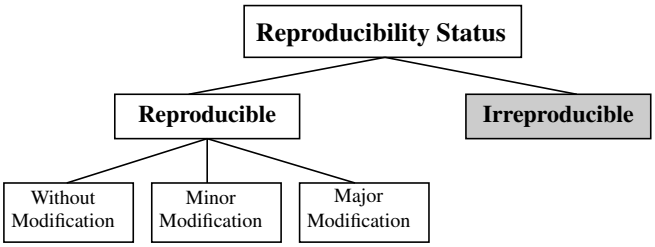


Fig. 6: Classification of issue reproducibility status

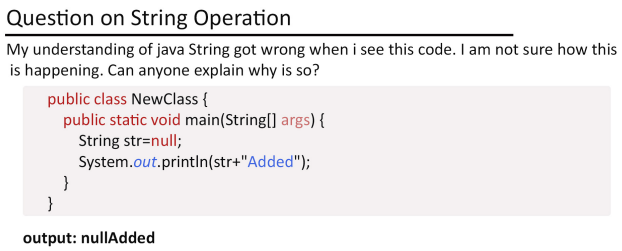


Fig. 7: An example question from Stack Overflow[8] (**Reproducibility status:** *reproducible without modification*)

two different dimensions. They are the success status of reproducibility and level of editing efforts in reproducing the issues reported at Stack Overflow questions. Reproducibility status can be classified into *two* major categories: *reproducible* and *irreproducible*. In the following section, we discuss both categories in detail.

- **Reproducible (REP)**: Reproducibility of issues usually requires some modifications to the code segments. In some cases, issues can be reproduced from the given code segments without any modification. We therefore classify the *reproducible* status into *three* more sub-categories based on the complexity of the code level modifications, the level of human efforts spent and the time required to reproduce the issue. Fig. 6 shows our sub-classification of the *reproducible* category.
  - *Reproducible without Modification (RWM):* The given code snippet is complete, stand-alone, and no explicit action is required to reproduce the issue. Fig. 7 shows a simple example of reproducible issue. Here, the user assigned *null* to a `String` object and then added to another string. Not surprisingly, he found that the *null* was also printed with the other string. To reproduce this issue, we just copy the code segment, paste it in the Eclipse IDE and then successfully reproduce the same output as noted in the question. Many of these issues are reported by the novice developers.
  - *Reproducible with Minor Modification (RMM):* The issue can be reproduced from the code by performing one or more modifications that are comparatively less complex and less time consuming. Box A shows the low cost modifications added to the code. We spend about 15–30 minutes to reproduce each programming issue from this sub-category.
  - *Reproducible with Major Modification (RMJM):* The reported issue can be reproduced from the code segment by performing one or more complex and time consuming modifications. Box B shows the high cost modifications. We spend about 30–60 minutes to reproduce each issue from this sub-category. Fig. 5 shows an example of reproducible issue where major modifications are performed on the code to reproduce the reported issue.
- **Irreproducible (IREP)** These issues are less likely to reproduce even after several minor and major code level modifications. As mentioned above, two of the authors took part in the manual analysis. When one fails to reproduce the issue, the same question is analysed by the other author. If both fail, we then conclude the issue as irreproducible. In some cases, we spent even a few hours for a single question. Fig. 1 shows an example where the issue could not be reproduced since important details of the code/implementation are missing. Table II shows our identified challenges that prevent a programming related issue from reproducing.

Box A : Minor Modifications

✛ Addition of a demo class or a main method or a method definition ✛ Creation of a constructor ✛ Declaration of an identifier/object whose type can be predicted easily. ✛ Initialization of an identifier/object by a default value or a sample value from the question ✛ Inclusion of the import statements (of native libraries) ✛ Handling an exception ✛ Resolve an external library dependency when the the code segment contains the import statements of them ✛ Resolve the less complex compilation errors ✛ Creation of the text/image/csv files ✛ Addition of sample values to the files from the question discussion ✛ Minor changes in the existing code ✛ Invoke a method with no parameters or known parameters.

Box B : Major Modifications

✛ Resolve external library dependency when the import statements are not in the code segment. ✛ Creation of database and table entries according to the problem description ✛ Major changes in existing code snippets ✛ Declaration of an identifier/object whose type cannot be predicted easily and their initialisation that demand code analysis ✛ Addition of sample input values to the files that are absent from question discussion ✛ Invoke a method with parameters where sample parameters are absent from the question description ✛ Code debugging

Besides the major classification above, we also consider the appropriateness of developers' claims about issues during the question submission. We found that there exist two more classes of issues that could not be reproduced as well. We discuss them in detail as follows:

StringTokenizer - first string?

I have this code to parse url string such as "?var=val" but when "search" is just "var=val" this code fails, how to make just "var=val" work as well?

```
StringTokenizer st1 =
new StringTokenizer(search, "?&;");
while(st1.hasMoreTokens()){
    String st2= st1.nextToken();
    int ii = st2.indexOf("=");
    if (ii > 0) {
        int ib = st2.length();
        myparms.put( st2.substring(0,ii) , st2.substring(ii+1,ib) );
    }
}
```

Fig. 8: An example question of Stack Overflow[9] with inaccurate claim about the programming issue

- *Inaccurate Claim (IAC):* In some cases, the stated programming issues in the Stack Overflow questions might not be accurate. We call these issues Inaccurate Claim. The question shown in the Fig. 8 shows an example of inaccurate claim about programming issue. Here, the user raised an issue that he could parse the string like `?var=val` but could not parse the string like `var=val`. When we attempt to execute this code in our IDE, we find that both strings can be parsed successfully. Even two programmers also commented as follows: *'Sorry, that code doesn't fail'* and *'This works for me too'*. We find similar occurrences with run-time errors, compiler errors, unexpected behaviour reported by the users at Stack Overflow questions. Such anomalies could have several explanations. First, this might happen due to the difference in the development environment. Second, users might fail to locate the defective code, and hence, they submit the wrong code fragments.
- *Ill-Defined Issue (IDEF)*: Ill-defined issues refer to the problems which cannot be reproduced consistently under any circumstances. That is, the question submitter does not specify the context precisely in which situation the alleged programming issue encounters. For instance, a user claimed that he was getting run-time exception after clicking buttons. During code execution we find several buttons in the user interface and some of them trigger runtime exceptions. However, since the user does not specify a button, the reported issue could also not be reproduced successfully.

B. Answering RQ_2

What proportion of reported issues at Stack Overflow questions can be reproduced successfully? We classify the issue reproducibility status into two major categories which are again classified into further sub-categories. We investigate the reproducibility of programming issues discussed at 400 randomly sampled questions from Stack Overflow. Fig. 9 shows the statistics on the reproducibility statuses of the questions. According to our analysis, 270 (67.50%) issues among 400 can be reproduced. On the contrary, 87 (21.75%) issues cannot be reproduced due to the challenges shown in Table II. We find that 7.25% issues are claimed inaccurately. This might happen due to the lack of programming experience or proper analysis of the code during question submission. Unfortunately, we could not determine the reproducibility of issues for 14 (3.50%) questions. The users often fail to provide the appropriate contexts in their questions which are required to reproduce the reported issues. We thus call them as ill-defined issues (IDEF).

[9]https://stackoverflow.com/questions/750557

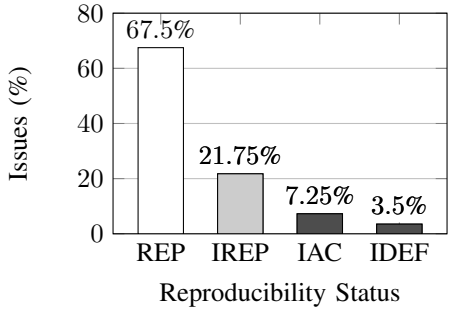


Fig. 9: Issue reproducibility status. **REP** = Reproducible, **IREP** = Irreproducible, **IAC** = Inaccurate Claim, **IDEF** = Ill-Defined Issue

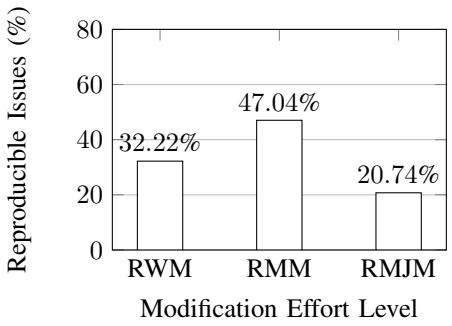


Fig. 10: Reproducible issues and modification effort level. **RWM** = Reproducible Without Modification, **RMM** = Reproducible with Minor Modification, **RMJM** = Reproducible with Major Modification

The ratio of reproducible status according to the effort level is shown in Fig. 10. A half of them require minor modification whereas 20.74% issues require major modification to make the code snippets capable of reproducing the reported issues. Fortunately 32.22% snippets can reproduce the issues without any modifications.

TABLE III
ISSUE REPRODUCIBILITY STATUS VS. PRESENCE OF ACCEPTED ANSWER (AA)

	AA Present	**AA Absent**	**Total**
Reproducible	201 (74.44%)	69 (25.56%)	270
Ireproducible	18 (20.69%)	69 (79.31%)	87
Total	219	138	357

C. Answering RQ_3

Reproduction of a programming issue discussed in the question texts allows one to experience the issue first hand. Such experience could help the users submit appropriate answers to the questions more promptly and more accurately. We thus analyze the relationship between reproducibility status of the

TABLE IV
ISSUE REPRODUCIBILITY SUBCATEGORY VS. PRESENCE OF ACCEPTED ANSWER (AA)

	AA Present	AA Absent	Total
Without Modification	66 (75.86%)	21 (24.14%)	87
Minor Modification	102 (80.31%)	25 (19.69%)	127
Major Modification	33 (41.07%)	23 (58.93%)	56
Ireproducible	18 (20.69%)	69 (79.31%)	87
Total	219	138	357

reported issues at Stack Overflow and the answer meta data such as presence of accepted answer, time delay between question and answer, and number of questions. In particular, we divide *RQ3* into three sub-questions, and answer them with detailed statistics as follows:

RQ$_3$(a): Does reproducibility of issues discussed at Stack Overflow questions encourage the acceptable answers? Table III shows the confusion matrix where the rows represent the reproducibility status (e.g., reproducible or irreproducible) and the columns represent the presence of accepted answer (e.g., present or absent). We note that 74.44% (201 out of 270) questions whose code segments are capable of reproducing the reported issues receive acceptable answers (i.e., solutions). On the contrary, only 20.69% (18 out of 87) questions with irreproducible issues receive the acceptable answers. Thus, the reproducibility of question issues increases the chance of getting a solution more than three times. We examine the correlation between the two categorical variables - reproducibility status and accepted answer presence. We use statistical test namely *Chi-Squared* test to measure the independence of these two categorical variables shown in Table III. We find statistically significant *p-value* ($p - value = 0 < 0.05$). Thus, there is a strong positive correlation between the issue reproducibility of questions and their chance of getting the acceptable answers.

We also further analyse the reproducibility status based on effort level and determine their correlation with the accepted answer presence as shown in Table IV. We again find statistically significant *p-value* ($p - value = 0 < 0.05$) from the *Chi-Squared* test. Thus, the human efforts in reproducing the question issues are significantly correlated with the chance of getting the accepted answer. From the Table IV we see that there is 75.86% (66 out of 87) chance of getting accepted answer when the issues can be directly reproduced from the verbatim code segment. On the contrary, the chance reduces to 58.93% (33 out of 56) when the submitted code needs major modifications. Surprisingly, the chance of getting accepted answer is highest (80.32%) for the questions whose code fragments require minor modifications to reproduce the issues. We thus further investigate this interesting scenario with more manual analysis. Three factors were identified that might explain such inconsistency:

- **Redundant and long code:** The users at Stack Overflow often submit questions that contain unnecessarily long code. Such questions often fail to get appropriate answers even though their code is capable to reproduce the reported issues without any modifications. According to our investigation, such code segments have more than 72 lines of code. We believe that such long code places extra burden on the other users during question answering. Thus, despite being reproducible without modifications, such code actually decreases the chance of getting an accepted answer.
- **Potential problem statements are not identified:** Users often fail to mention the potential problematic statements within the submitted code in their questions. This increases the analysis time for the other users at Stack Overflow while answering the question. Thus, despite the verbatim code segments being able to reproduce the question issues (without modifications), they might not receive the accepted answers.
- **Novice user:** Sometimes novice users add long code (e.g., an entire course assignment) without proper analysis. They even do not discuss about the issues properly. According to our analysis, 50% users who added long and redundant code and did not get an exact answer have a reputation score below 20.

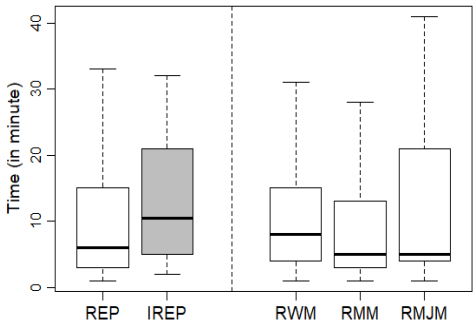


Fig. 11: Time spent for reproducible and irreproducible issues

RQ$_3$(b): Does reproducibility of issues discussed at Stack Overflow questions reduce the time delay of getting the accepted answers? According to RQ3a, there is a strong correlation between reproducibility status of question issues and the chance of getting an accepted answer. In fact, the chance is more than three times higher for the reproducible issues. In this section, we investigate whether the delay of getting the accepted answer could be influenced by the reproducibility status of the submitted issues at Stack Overflow questions. We determine the delay between the submission time of a question and that of the accepted answer, and contrast between reproducible and irreproducible issues using such delays. Fig. 11 shows the box plots for the delay of getting the accepted answers. We see that the median delay of getting an accepted answer is about 5 minutes when the issue reported at the question is reproducible. On the contrary, such delay is almost double when the reported issue is not reproducible with the submitted code. We also find a significant difference in time delay for getting the accepted answer between reproducible and irreproducible status. We use *Mann-Whitney-Wilcoxon* test, a non-parametric statistical significance test and get statistically significant *p-value* (p-value $= 0.04 < 0.05$). We also examine the effect size using *Cliff's delta* test and find

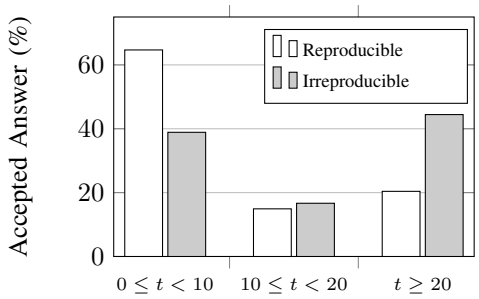


Fig. 12: Accepted answer fraction vs. the time delay between question and accepted answer submission

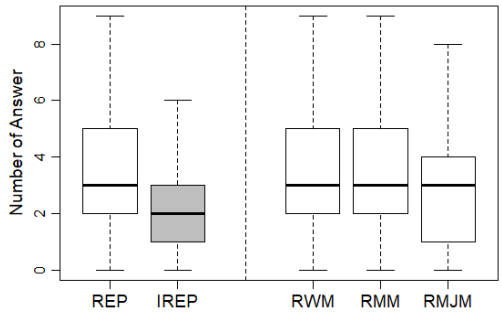


Fig. 13: Number of answers for the questions with reproducible and irreproducible issues

large *effect size*, i.e., Cliff's $|d| = 0.9365857$ (large) with 95 percent confidence. Given all these evidences above, the delay of getting an acceptable answer is significantly higher for the questions with irreproducible issues. Thus, reproducibility of the reported issues is an important quality paradigm for Stack Overflow questions, and such attribute could help them get the accepted answers quickly, even within 5 minutes.

TABLE V
REPRODUCIBILITY STATUS VS. TIME DELAY BETWEEN QUESTION AND ACCEPTED ANSWER SUBMISSION

	$t < 10$	$t \geq 10$	**Total**
Reproducible	130 (64.68%)	71 (35.32%)	201
Ireproducible	7 (38.89%)	11 (61.11%)	18
Total	137	82	219

Although the above box plots demonstrate the benefits of issue reproducibility, we further classify the delay of getting accepted answers into three intervals: $0 \leq t < 10$, $10 \leq t < 20$, $t \geq 20$. Fig. 12 shows the percentage of the accepted answers for reproducible and irreproducible issues against these intervals. We see that questions with reproducible issues receive about 65% of their accepted answers within only 10 minutes. Such percentage is only 39% for the questions with irreproducible issues. It also should be noted that 44% of these questions require more than 20 minutes on average to receive the accepted answers. Table V provides the raw statistics on the accepted answers and time interval. We examine the relation between these two categorical variables using statistical test namely *Chi-Squared* test. Thus, given all the evidences in above the tables and plots, issue reproducibility status is very likely to influence the time delay for getting the acceptable answers at Stack Overflow.

RQ_3(c): Does reproducibility of issues discussed at Stack Overflow questions encourage more answers? According to RQ3a and RQ3b, the reproducibility of reported issues at Stack Overflow question might encourage quick and high quality responses from the users. In this section, we further investigate whether such reproducibility also encourages more answers at Stack Overflow. Fig. 13 shows the box plots for the answer count of our selected questions against their issue reproducibility status. We clearly see that questions with issue reproducibility receive more answers on average than the counterpart. We also find a significant difference in the number of answers between two types of questions. *Mann-Whitney-Wilcoxon* test shows $p-value = 0$ whereas *Cliff's* $|d|$ *= 0.57 (large)* with 95% confidence. Given these statistical findings, we suggest that the reproducibility status of a reported issue at Stack Overflow question has a significant impact on its chance of getting more answers.

V. KEY FINDINGS & GUIDELINES

While submitting a question at Stack Overflow, it is recommended to add a bit of code fragment so that the reported program issues can be reproduced easily [22]. Experts users of Stack Overflow also suggest to add complete and standalone code in the question [20]. However, our study delves further into the submitted code, and delivers more in-depth insights on the question issue reproducibility using such code.

(a) **Redundancy Hurts:** Only those statements should be added that are required to reproduce the reported issue and the redundant code should be avoided. Long and redundant code waste the developers time unnecessarily which might also hurt the question's chance of getting an accepted answer.

(b) **Dependency Matters:** Adding the import statements targeting the external libraries is very important. This is one of the major difficulties that we faced during the reproduction of question issues with the submitted code. The question texts also should point to the external libraries (if used) so that the users can include them in the IDE during issue reproduction.

(c) **Executable Code for Debugging:** The submitted code segment should compile and run if the reported issue requires debugging to reproduce. This is especially needed when the program shows stochastic or unexpected behaviours. Without a reproducible code, such questions are generally hard to answer effectively.

(d) **One Issue Per Question:** Multiple issues should not be discussed using a single code segment. Separate questions and code snippets are encouraged for separate programming issues for effective question answering.

VI. THREATS TO VALIDITY

Threats to *external validity* relate to the generalizablity of a technique. We manually analyse a limited number of questions and as such our results may not generalise to all the

questions. However, we investigate a wide variety of questions of different types of issues in order to combat potential bias in our results. Nonetheless, replication of our study using additional questions and different languages may prove fruitful. Besides, we investigate only Java code segments. However, we believe that our insights can generalise to other statically-typed, compiled programming languages such as C++ and C#. But we caution readers to not over-generalise our results.

Threats to *internal validity* relate to experimental errors and biases [23]. The reproducibility status (e.g., reproducible, irreproducible) of the reported issue is threatened by the subjectivity of our classification approach. Thus, we cross-validate our results when an issue cannot be reproduced. We finalise the reproducibility status as irreproducible if we both fail to reproduce the question issue.

Threats to *construct validity* relate to suitability of evaluation metrics. We use *Mann-Whitney-Wilcoxon* test which is a widely used non-parametric test for evaluating the difference between two sample sets. However, the significance level might suffer due to the limited size of the samples. We thus consider the effect size along with the *p-value*. To see the correlation between two categorical variables we use *Chi-square* test. This statistical test of independence works well when there is a small number of categories (≤ 20) [13].

VII. Related Work

There have been several studies on the reproducibility of software bugs [10, 14, 21, 28]. However, to the best of our knowledge, ours is the first work that investigates the reproducibility of the reported issues at Stack Overflow questions using their submitted code.

Yang et al. [29] and Horton and Parnin [11] are the only closely related studies in terms of research methodologies and problem aspects. Yang et al. analyse the *usability* of 914,974 Java code snippets on Stack Overflow and report that only 3.89% are parsable and 1.00% are compilable. They analyse the code segments found in only the accepted answers of Stack Overflow and employ automated tool such as Eclipse JDT and ASTParser for the parsing and compiling the code. Then they report the errors that prevent the code segments from parsing and compiling without human involvement. Similarly, we analyse the Java code snippets from the questions of Stack Overflow. However, unlike their approach which is completely automated, our approach is a combination of automatic and manual analysis. Not only we make the code parsable, compilable and runnable using appropriate modifications to the code, we also overcome the challenges to make them reproduce the reported issues at Stack Overflow.

Horton and Parnin investigate the executability of Python code found on GitHub Gist system. Their primary focus was the execution of the Python snippets. However, we go beyond code execution and manually investigate the reproducibility of issues using Java code snippets submitted with Stack Overflow questions. They also report the types of execution failures encountered while running Python gists. Similarly, we categorise the reproducibility status and identify the reasons why the issues could not be reproduced. Interestingly some reasons are common between ours and their study such as import error, syntax error. However, executability of a code does not always guarantee the reproducibility of an issue reported at Stack Overflow question. Reproducibility requires testing and debugging which warrant for manual analysis, and this was not done by any of the earlier works. Besides, our research context differs from theirs since they examine gists shared on GitHub whereas we deal with code snippets found in Stack Overflow questions.

Due to the growing popularity and importance of Stack Overflow Q&A site, there have been several studies that focus on question/answer quality analysis. Duijn et al. [9] collect the Java code segments found in Stack Overflow questions and suggest that several code level constructs (e.g., code length, keywords) are correlated to the quality of a code fragment. A number of studies investigate the quality of a code segment by measuring its readability [4, 5, 8, 16, 18, 24] and understandability [12, 19, 26]. Unfortunately, their capability of reproducing the issues reported at Stack Overflow questions was not investigated by any of the early studies. This makes our work novel. As our empirical findings suggest, like readability and understandability, the reproducibility of issues could be considered as one of the major quality aspects of Stack Overflow questions.

Several other studies [6, 7, 22, 27] investigate how to get a fast answer, create a high quality post or mine a successful answer. They suggest that information presentation, code-text ratio and question posting time are the key factors behind getting the high quality answers. Similarly, our findings show that reproducibility of an issue discussed in the question is likely to encourage high quality responses including the acceptable answers. Rahman and Roy [17] investigate why questions at Stack Overflow remain unresolved. However, they also do not consider the issue reproducibility in their study.

VIII. Conclusion and Future Work

In this paper, we manually analyse 400 randomly selected questions from Stack Overflow and investigate the reproducibility of their reported issues using their code segments. We answer three research questions in this work. We classify status of reproducibility into two major categories - reproducible and irreproducible. We find that 68% of issues can be reproduced using the submitted code after performing minor or major modifications to them. We also investigate and report why several issues could not be reproduced using the attached code segments. We then examine the correlation between the reproducibility of issues (of questions) and answer meta-data. Our findings suggest that reproducibility of question issues is likely to encourage more high quality responses including the acceptable answers. Thus, reproducibility of issues can also be treated as a novel metric of question quality for Stack Overflow which was ignored by the earlier studies. In future, we plan to develop automated tool supports for predicting the reproducibility of a given question issue and for turning irreproducible issues into reproducible ones.

REFERENCES

[1] SO question issue reproducibility: Replication package. URL https://bit.ly/2JqxabU.

[2] StackExchage API. URL http://data.stackexchange.com/stackoverflow.

[3] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A Schneider. Answering questions about unanswered questions of stack overflow. In *Proc. MSR*, pages 97–100, 2013.

[4] R. PL Buse and W. R. Weimer. A metric for software readability. In *Proc. ISSTA*, pages 121–130, 2008.

[5] R. PL Buse and W. R. Weimer. Learning a metric for code readability. *TSE*, 36 (4):546–558, 2010.

[6] F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli. Mining successful answers in stack overflow. In *Proc. MSR*, pages 430–433, 2015.

[7] F. Calefato, F. Lanubile, and N. Novielli. How to ask for technical help? evidence-based guidelines for writing questions on stack overflow. *IST*, 94:186–207, 2018.

[8] E. Daka, J. Campos, G. Fraser, J. Dorn, and W. Weimer. Modeling readability to improve unit tests. In *Proc. FSE*, pages 107–118, 2015.

[9] M. Duijn, A. Kucera, and A. Bacchelli. Quality questions need quality code: Classifying code fragments on stack overflow. In *Proc. MSR*, pages 410–413, 2015.

[10] M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso. Automatically translating bug reports into test cases for mobile apps. In *Proc. ISSTA*, pages 141–152, 2018.

[11] E. Horton and C. Parnin. Gistable: Evaluating the executability of python code snippets on github. In *Proc. ICSME*, pages 217–227, 2018.

[12] JC Lin and KC Wu. Evaluation of software understandability based on fuzzy matrix. In *Proc. WCCI*, pages 887–892, 2008.

[13] M. L. McHugh. The chi-square test of independence. *BM*, 23(2):143–149, 2013.

[14] K. Moran, M. Linares-Vsquez, C. Bernal-Crdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *Proc. ICST*, pages 33–44, 2016.

[15] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza. Understanding and classifying the quality of technical forum questions. In *Proc. QSIC*, pages 343–352, 2014.

[16] D. Posnett, A. Hindle, and P. Devanbu. A simpler model of software readability. In *Proc. MSR*, pages 73–82, 2011.

[17] M. M. Rahman and C. K. Roy. An insight into the unresolved questions at stack overflow. In *Proc. MSR*, pages 426–429, 2015.

[18] S. Scalabrino, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Improving code readability models with textual features. In *Proc. ICPC*, pages 1–10, 2016.

[19] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically assessing code understandability: how far are we? In *Proc. ASE*, pages 417–427, 2017.

[20] J. Skeet. The golden rule: Imagine youre trying to answer the question, 2010. URL https://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question.

[21] M. Soltani, A. Panichella, and A. van Deursen. A guided genetic algorithm for automated crash reproduction. In *Proc. ICSE*, pages 209–220, 2017.

[22] M. Squire and C. Funkhouser. " A bit of code": How the stack overflow community creates quality postings. In *Proc. HICSS*, pages 1425–1434, 2014.

[23] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Proc. CSMR-WCRE*, pages 44–53, 2014.

[24] C. Treude and M. P. Robillard. Understanding stack overflow code fragments. In *Proc. ICSME*, pages 509–513, 2017.

[25] C. Treude, O. Barzilay, and M. Storey. How do programmers ask and answer questions on the web?: Nier track. In *Proc. ICSE*, pages 804–807, 2011.

[26] A. Trockman, K. Cates, M. Mozina, T. Nguyen, C. Kästner, and B. Vasilescu. Automatically assessing code understandability reanalyzed: combined metrics matter. In *Proc. MSR*, pages 314–318, 2018.

[27] S. Wang, TH Chen, and A. E Hassan. Understanding the factors for fast answers in technical q&a websites. *ESE*, 23(3):1552–1593, 2018.

[28] M. White, M. Linares-Vásquez, P. Johnson, C. Bernal-Cárdenas, and D. Poshyvanyk. Generating reproducible and replayable bug reports from android application crashes. In *Proc. ICPC*, pages 48–59, 2015.

[29] D. Yang, A. Hussain, and C. V. Lopes. From query to usable code: an analysis of stack overflow code snippets. In *Proc. MSR*, pages 391–402, 2016.