Reproducibility Challenges and Their Impacts on Technical Q&A Websites: The Practitioners' Perspectives

Saikat Mondal
University of Saskatchewan, Canada
saikat.mondal@usask.ca

Banani Roy
University of Saskatchewan, Canada
banani.roy@usask.ca

ABSTRACT

Software developers often look for solutions to their code-level problems by submitting questions to technical Q&A websites like Stack Overflow (SO). They usually include example code segments with questions to describe the programming issues. SO users prefer to reproduce the reported issues using the given code segments when they attempt to answer the questions. Unfortunately, such code segments could not always reproduce the issues due to several unmet challenges (e.g., external library not found) that might prevent questions from receiving prompt and appropriate solutions. A previous study produced a catalog of potential challenges that hinder the reproducibility of issues reported at SO questions. However, it is unknown how the practitioners (i.e., developers) perceive the challenge catalog. Understanding the developers' perspective is inevitable to introduce interactive tool support that promotes reproducibility. We thus attempt to understand developers' perspectives by surveying 53 users of SO. In particular, we attempt to – (1) see developers' viewpoints on the agreement to those challenges, (2) find the potential impact of those challenges, (3) see how developers address them, and (4) determine and prioritize tool support needs. Survey results show that about 90% of participants agree to the already exposed challenges. However, they report some additional challenges (e.g., error log missing) that might prevent reproducibility. According to the participants, too short code segment and absence of required Class/Interface/Method from code segments severely prevent reproducibility, followed by missing important part of code. To promote reproducibility, participants strongly recommend introducing tool support that interacts with question submitters with suggestions for improving the code segments if the given code segments fail to reproduce the issues.

CCS CONCEPTS

• **Software and its engineering** → *Software repository mining*; *Software maintenance and evolution*; *Practitioners' perspective*; *Software maintenance tools*.

KEYWORDS

Stack Overflow, issue reproducibility, code segments, reproducibility challenges, user study

1 INTRODUCTION

Stack Overflow (SO) has emerged as one of the largest and most popular technical question and answer (Q&A) sites. SO is continuously contributing to the body of knowledge in software development [33, 36, 40]. Millions of software developers interact through SO every month to solve their programming-related problems [11]. About seven thousand questions are submitted in SO every day. Among them, a large number of questions discuss code level problems (e.g., errors, unexpected behaviour) [36]. Such questions often include problematic code segments with the programming issue descriptions. SO users generally prefer to reproduce the issues reported at the questions using the code segments and then submit their solutions [18]. Reproducibility means a complete agreement between the reported and investigated issues [18, 41]. Unfortunately, such programming issues could not always be reproduced by other users due to several unmet challenges (e.g., too short code segment) of the code segments [18, 19, 32]. This phenomenon prevents the questions from getting prompt and appropriate solutions. Mondal et al. [18] show that a question whose code segment could reproduce the reported issue has more than three times higher chance of receiving an acceptable answer (i.e., working solution) than the question whose code segment could not reproduce the issue. Moreover, the median time delay of receiving an accepted answer is double, and the average number of answers is significantly less for the questions with irreproducible issues than those of the questions with reproducible issues.

Table 1: Challenges preventing issue reproduction

• Class/Interface/Method not found • Important part of code missing • External library not found • Identifier/Object type not found • Too short code snippet • Database/File/UI dependency • Outdated code

A couple of existing studies [13, 43] investigate the challenges of usability and executability of the code segments posted on Q&A sites. For example, a study conducted by Yang et al. [43] analyzes the parsability and compilability of code segments extracted from the accepted answers of SO. However, their analysis was fully automatic that only exposes parse and compile errors (e.g., syntax errors, incompatible types). Horton and Parnin [13] examine the

executability of the Python code segments found on the GitHub Gist system. They identify several flaws (e.g., syntax errors, indentation errors) that prevent the executability of such code segments. However, a simple execution success does not always guarantee the reproducibility of issues [18]. Several studies investigate the quality of SO code segments by measuring their readability [4–6, 26, 30, 37], and understandability [17, 29, 38]. Unfortunately, their capability of reproducing the issues reported at SO questions was not investigated. Thus, their approach also fails to address reproducibility challenges. Mondal et al. [18] first investigate the reproducibility of issues reported at SO questions related to Java programming language. Their investigation produces a catalog of challenges (see Table 1) that might prevent reproducibility. However, the catalog was not validated by the practitioners (i.e., developers). Thus, it is unknown to us how the developers perceive the reproducibility challenges. Understanding the developers' perspective is important to introduce efficient tool support to promote reproducibility.

This study addresses developers' perspectives on the reproducibility challenges and their impact by surveying 53 software developers. In particular, we attempt to – (1) validate the challenge catalog (see Table 1) by asking developers' agreement to those challenges, (2) understand the potential impact of each of those challenges to answer SO questions, (3) see how developers address reproducibility challenges, and (4) find developers' recommendations on designing efficient tool support to help question submitters improve their code segments to promote reproducibility.

Replication Package that contains the survey questionnaire and all the responses is shared in our online appendix [2].

2   BACKGROUND AND RELATED WORK

In this section, we start by introducing the issue reproducibility and the challenges that prevent reproducibility. We then proceed to describe related studies.

Figure 1: An example [25] question of SO that discusses a programming issue.

2.1   Issue Reproducibility

Reproducibility is often closely related to repeatability and replicability [1]. However, its definition differs across disciplines. In this study, reproducibility means a complete agreement between the reported and the investigated issues [18, 41]. Consider the example question in Fig. 1, where a user was trying to take a person's full name (e.g., John Doe) as input by invoking the `next()` method

of Java `Scanner` class. However, when the user was printing the name, only the first name (e.g., John) was being printed, and the last name (e.g., Doe) was getting lost. In particular, the user was not getting the part of the name after space. She included the definition of the method `processName()` with the question description, where she was taking and printing the name. If other users also find the first part of the name and lose the part after space by invoking `processName()` method, that means the issue is reproducible. On the contrary, when others fail to regenerate the reported issue using the given method, it suggests the issue is not reproducible.

2.2   Reproducibility Challenges

Mondal et al. [18] analyze 400 SO questions related to the Java programming language and attempt to reproduce their issues. However, they could not reproduce about 22% issues due to several unmet challenges of the code segments. Table 1 shows those reproducibility challenges. In this section, we discuss the challenges that prevent reproducibility as follows.

Figure 2: An example [21] question of SO whose issue could not be reproduced due to mainly two unmet challenges – (i) class/interface/method not found and (ii) important part of code missing.

(1) **Class/Interface/Method not found.** Developers often submit only the code segments of interest with questions, which are neither complete nor compilable. Such code segments also invoke methods from classes. However, the code segments often miss the definition of the methods. Let us consider the example question in Fig. 2, where the developer attempts to reset all the variables by the default values. However, the code was not working as expected, and some of the variables were retaining their old values. Unfortunately, this issue could not be reproduced due to several unmet challenges. Especially, definitions of the `Game` class and `playGame()` method are missed but essential to reproduce the issue. Thus, one developer commented while attempting to answer the question – *"Can you post more code? The Game class? The class that contains the restart() method?"*.

(2) **Important part of code missing.** Code segments included with questions often miss such statements (i.e., part of code) without which issues could not be reproduced. Here, "an important part" means such part of code that could never be guessed properly. For instance, we could add the definition of `Game` class and `playGame()` method in the code example in Fig. 2 and make the code executable. However, we cannot reproduce the issue. We need the part of the code inside the class and the method by which the developer attempted to reset the variables.

Getting Error On Xml resultSet in java

when I try run to execute java code it displays error. which I used to get xml type resultset from oracle using following method in java

```
poxml = XMLType(rs.getOPAQUE(1));
```

it shows cannot find symbol --- symbol : method getOPAQUE(int)

any one can suggest any other method for retrieving the xml resultset values from oracle

`java`

Figure 3: An example [24] question of SO whose issue could not be reproduced due to mainly three unmet challenges – (i) external library not found, (ii) identifier/object type not found and (iii) too short code snippet.

(3) **External library not found.** Resolving external library dependencies is one of the major challenges to reproduce the issues from the submitted code segments [18]. Java has thousands of external (i.e., third-party) libraries with millions of classes and methods. However, different libraries contain classes and methods with the same name. Thus, if the code segments miss import statements of the external libraries or question descriptions do not have hints that point to the appropriate libraries, developers face difficulty adding the appropriate libraries. As a result, they could not compile/execute the code and fail to reproduce the question issues. Consider the code example shown in Fig. 3, where the import statement for the library that contains class `XMLType` was missing. Thus, the inclusion of the appropriate external library is a major challenge here to reproduce the question issue.

(4) **Identifier/Object type not found.** Code segments use identifiers/objects without declaring them. In some cases, developers infer the type of identifiers/objects by looking at the assigned values or method invocations. Otherwise, they find difficulties in inferring their types that might prevent the reproducibility of issues. Consider the code example in Fig. 3, where the types of `poxml` and `rs` are unknown and hard to guess from the submitted code.

(5) **Too short code snippet.** Developers often submit incomplete code segments with their questions. However, code segments are too short to reproduce the question issues in many cases. For example, Fig. 3 shows a question where the question submitter included only one line of code with the issue description. Thus, it is genuinely challenging to guess the missing statements and make the code compilable/executable to reproduce the issue. The challenge "too short code segment" could overlap with an "important part of code missing". However, an important part of code could be missed even when the developers submit a long code.

(6) **Database/File/UI dependency.** Several code segments could not reproduce the issues due to their complex interactions with databases, external files, and UI elements. Consider the example question in Fig. 4, where the developer attempted to create and load an XML doc. Unfortunately, the code got a null exception when the developer attempted to access the doc more than once. Such an issue could not be reproduced due to the external file dependency. However, the doc could not be created because the question lacks important detail (e.g., file path, content) associated with the file.

(7) **Outdated code.** A few code segments contain outdated code (e.g., deprecated class/API) that could prevent issue reproducibility.

get resource is null

I have an @Test method with invocationCount=3. Each time this method is run, there is a call for preparing some doc into another method. All it's working GREAT when the @Test is run for the 1st time. The doc is successfully found and loaded:

```
final DocumentRegistryResource documentRegistryResource =
    RestClientFactory.getInstance().createDocumentRegistryResource(
    getUserRestAuth());
final File importFile = new File(
    this.getClass().getResource("/documents-template-test.xml").getFile());
```

BUT, at the 2nd and 3rd invocation, I receive a null exception:

this.getClass().getResource("/documents-template-test.xml") is no longer found.

`java`

Figure 4: An example [23] question of SO whose issue could not be reproduced due to mainly two unmet challenges – (i) database/-file/UI dependency and (ii) class/interface/method not found.

Use Sounds in java

When I create an object from below class and send the address of audio to its constructor, I have to send the complete address like:

"C:\Documents and Settings\TheHero\My Documents\NetBeansProjects\project1\src\project1\s1.wav"

But I want to use this code just with the name of sound like: "s1.wav".

I copied the sound into the src folder and then use "s1.wav" and the result was "java.io.FileNotFoundException". If I use the complete address of sound when I run the program on another computer, it does not work. I'm right?

```
public class MusicPlayer{
    private String filename;
    public MusicPlayer(String filename) {
        this.filename = filename;
    }
    public void play() {
        try {
            InputStream in = new FileInputStream(filename);
            AudioStream as = new AudioStream(in);
            AudioPlayer.player.start(as);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

`java`

Figure 5: An example [22] question of SO whose issue could not be reproduced due to mainly outdated code challenge.

A few studies investigate the outdated code of Stack Overflow [27, 44]. However, in this study, we consider a code is outdated when the code contains such classes/APIs that are not compatible with JDK-1.8. It is hard to find an equivalent or alternative class/API for the outdated class/API in some cases. Consider the code segment in Fig. 5, where the classes `AudioStream` and `AudioPlayer` are not being used in JDK-1.8. Thus, the developers face difficulties in finding their alternatives while reproducing the question issue.

2.3 Related Work

Several studies investigate usability (e.g., parsability, compilability) [43], executability [13] and reproducibility [9, 13, 18–20, 28, 32, 41, 43] challenges of the code segments posted at crowd-sourced developer forums (e.g., GitHub, SO). However, our study first investigates the practitioners' perspective on the reproducibility challenges of the programming issues reported at SO questions, their impact to answer questions, and interactive tool design requirements to promote reproducibility.

Yang et al. [43] analyze the usability of about 914K Java code segments extracted from the accepted answers of SO. They expose several challenges of their parsability (e.g., syntax error) and compilability (e.g., incompatible types). The authors employ automated tools such as Eclipse JDT and ASTParser to parse and compile the code segments and report the challenges that prevent them from parsing and compiling. However, they do not analyze the challenges of reproducibility. Horton and Parnin [13] investigate the executability of Python code found on the GitHub Gist system. They report the types of execution failures encountered while running Python gists such as import error, syntax error, indentation error. However, a code segment's execution success does not always guarantee the reproducibility of an issue reported at the SO question. Reproducibility may require testing and debugging that warrant manual analysis, which was not done by Horton and Parnin [13]. Mondal et al. [18] go beyond code execution and manually investigate the reproducibility of issues reported at 400 questions related to Java programming language using the code segments included with questions. They produce a catalog of challenges that prevent reproducibility of SO question issues. However, it is important to listen to the practitioners whether they agree to the challenges and understand the potential impact of those challenges to answer a question that was not done by any earlier studies.

Several researchers investigate the reproducibility challenges of software bugs and security vulnerabilities [8, 20, 28]. Joorabchi et al. [8] analyze 1,643 irreproducible bug reports and investigate the causes of their irreproducibility. They reveal six root causes, such as environmental differences, insufficient information that prevent bug-reports' reproducibility. Rahman et al. [28] conduct a study to understand the irreproducibility of software bugs. They investigate 576 irreproducible bug reports from two popular software systems (e.g., Firefox, Eclipse) and identify 11 challenges (e.g., bug duplication, missing information, ambiguous specifications) that might prevent bug reproducibility. The authors then survey 13 developers to understand how the developers cope with irreproducible bugs. According to the study findings, developers either close these bugs or solicit further information. Mu et al. [20] analyze 368 security vulnerabilities to quantify their reproducibility. Their study suggests that individual vulnerability reports are insufficient to reproduce the reported vulnerabilities due to missing information. Besides, many vulnerability reports do not include details of software installation options and configurations, or the affected operating system (OS) that could hinder reproducibility. Then they survey hackers, researchers, and engineers who have domain expertise in software security. Survey findings suggest that apart from internet-scale crowd-sourcing and some interesting heuristics, manual efforts (e.g., debugging) based on experience are the sole way to retrieve missing information from reports. Our study relates to the above studies in terms of research methodologies and problem aspects. However, our research context differs from theirs since we attempt to understand the practitioners' perspectives on SO questions' reproducibility issues.

Tahaei et al. [33] analyze 1,733 privacy-related questions of SO to understand the challenges and confusion that developers face while dealing with privacy-related topics. Ebert et al. [7] investigate the reasons (e.g., missing rationale) and impacts (e.g., merge decision is delayed) of confusion in code reviews. Their study suggests how developers cope with confusion during code reviews. For example, developers attempt to deal with confusion by requesting information, improving the familiarity with existing code, and discussing off-line the code review tool. They survey developers to obtain actionable insights for both researchers and tool builders. While our work overlaps with them in terms of methodology, however, our research goals are different. They attempt to find reasons and impacts of confusion in code reviews. However, we survey 53 developers to understand (1) the impacts of reproducibility of SO questions issues, (2) how developers plan to modify the code segments to make them capable of reproducing the issues, and (3) the insights for introducing interactive tool supports.

Ford et al. [10] deploy a month-long, just-in-time mentorship program to SO to guide the novices with formative feedback of their questions. Such mentorship reduces the negative experience caused by delays in getting answers or adverse feedback. However, human mentorship is costly. Thus, it is hard to sustain such mentorship. Horton and Parnin [14] present DockerizeMe, a system for inferring the dependencies required to execute a Python code snippet without import errors. Terragni et al. [34] propose a technique CSnippEx to automatically convert Java code segments into compilable Java source code files. However, we plan to determine the design requirement to introduce an intelligent and interactive tool based on the developers' recommendations. Such a tool could analyze SO code segments included with questions related to Java programming problems and suggest users improve the code segments to promote reproducibility.

3 STUDY DESIGN

3.1 Objective and Research Questions

In this study, we aim to understand developers' perspectives on reproducibility challenges and estimate their impacts to answer SO questions. We also plan to introduce tool support to promote reproducibility by seeking developers' suggestions on the design of the tool. Following this aim, we guide our study with five research questions and make five contributions in this paper as follows.

• **(RQ₁) What do developers consider to be the challenges behind the irreproducibility of issues reported at Stack Overflow questions?** Mondal et al. [18] produce a catalog of challenges (see Table 1) that might prevent reproducibility. However, developers' feedback on such empirical findings is essential to increase confidence in the findings [28]. To answer RQ₁, we present four example questions of SO with their issue descriptions and code segments. Participants were asked to reproduce the reported issues using the code segments. Upon failure, we ask for their agreement to the given reproducibility challenges. The given challenges were validated by 53 software developers with an agreement level between 80% and 94%.

• **(RQ₂) What are the perceived impacts of the reproducibility challenges to answer a Stack Overflow question?** Understanding the impact of each of the challenges is important to determine which challenges must be resolved by question submitters to receive appropriate solutions to their questions. To answer RQ₂, we present each of the challenges with five options – (1) *not a problem*, (2) *moderate*, (3) *severe*, (4) *blocker*, and (5) *no opinion*. We then ask participants to select one of the five options. Participants

assessed "an important part of code missing" as mostly a blocker, and "outdated code" as mostly not a problem.

• **(RQ₃) How do developers prioritize to address the reproducibility challenges?** Prioritization of challenges is important to determine which challenges need to be fixed first within a strict budget (e.g., time constraint). We thus ask participants to specify three reproducibility challenges they would like to prioritize above others. We find the following challenges in the top three list – (1) an important part of code missing, (2) class/interface/method not found, and (3) too short code snippet.

• **(RQ₄) How do developers plan to modify the code segments to make them capable of reproducing the question issues?** Developers' code editing plan to reproduce the question issues offers more insights into how an intelligent tool could suggest question submitters improve their code segments. We thus attempt to observe participants' editing actions to address the reproducibility challenges. We see that participants perform several editing actions such as adding demo classes and methods, declaring and initializing variables, invoking methods, including external and native libraries. In particular, the participants first attempt to make the code segments compilable/executable to reproduce the issues. On the contrary, several participants do not modify the code segments. They consider the code segments insufficient to make them compilable/executable, or modifications based on assumptions (without appropriate hints) could deviate the code segments much from the original code.

• **(RQ₅) What are the interactive tool design requirements to address the reproducibility challenges?** Questions with reproducible issues have a significantly higher chance of receiving acceptable answers with minimum time delay [18]. However, the current question submission system of SO is not capable of addressing reproducibility issues. To introduce tool support, we thus ask participants' recommendations on tool design to promote reproducibility. According to them, intelligent tools that (1) analyze the code segments statically to find the reproducibility challenges and (2) suggest question submitters improving the code examples could help them receiving appropriate solutions to their questions.

3.2 Methodology

In the following, we describe the study methodology. First, we survey to understand *"what developers say"* (**RQ₁**, **RQ₂**, and **RQ₃**) about reproducibility challenges and their potential impact to answer questions. Then we see the code modification plan to understand *"what developers do"* (**RQ₄**) to reproduce the issues. Next, we find *"what developers suggest"* (**RQ₅**) to receive recommendations that inform the design needs of the interactive tool supports to assist reproducibility of question issues. Finally, we analyze the survey findings and report them.

3.2.1 Survey. We conduct an online survey aiming at validating and extending the catalog of reproducibility challenges (see Table 1) identified by Mondal et al. [18]. Furthermore, we ask the participants to give their viewpoints on the impacts of those challenges and their potential fixes. We primarily follow Kitchenham and Pfleeger's guidelines for personal opinion surveys [16]. However, we also consider the guidance and ethical issues from the established best practices [12, 31].
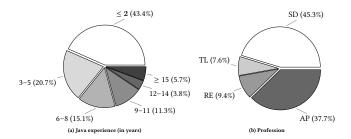
Figure 6: Information of the survey participants (SD: Software Developer, TL: Technical Lead, RE: Research Engineer, AP: Academic Practitioner).

Survey Design. Our survey includes different types of questions (e.g., multiple-choice and free-text answers). Before asking questions, we explain the purpose of the survey and our research goals to the participants. We ensure the survey participants that the information they provide must be treated confidentially. We first piloted the preliminary survey with a small set of practitioners (i.e., 3 participants). We collect their feedback on (1) whether the length of the survey was appropriate and (2) the clarity and understandability of the terms. We then perform minor modifications to the survey draft based on the received feedback and produce a final version. We inform the estimated time (i.e., 45–50 minutes) required to complete the survey to the participants based on the pilot survey. We exclude the three responses from the pilot survey from the presented results in this paper. Our survey comprises five parts as follows.

(i) *Consent and Prerequisite.* In this part, we ask the participants to confirm whether they consent to participate in this survey and agree to process their data. We also ask questions to confirm whether they answer SO questions and have experience in Object-Oriented Programming (OOP), especially Java. Otherwise, we did not allow them to participate in our survey.

(ii) *Participants Information.* In this part, we attempt to collect some information about the participants, such as years of experience in OOP and professional software development, their current profession, SO account age, and their question answering experience in SO.

(iii) *Agreement to the Reproducibility Challenges.* In this part, we present four SO questions from the manually analyzed dataset by Mondal et al. [18]. We ask participants to reproduce the question issues using the code segments. Upon failure, we ask their agreement/disagreement with the challenges. Otherwise, we ask for their code editing actions required to perform to reproduce the issues. In both cases, there are options to report additional challenges or editing actions.

(iv) *Impact of the Reproducibility Challenges.* Here, we ask questions to understand the potential impact and severity of each of the reproducibility challenges.

(v) *Tool Support Needs.* Finally, we seek the participants' opinions on the needs and design requirements of an intelligent tool to promote reproducibility. We also offer a few tool support options and employing a 5-point Likert Scale [15, 39] to estimate the participants' responses.

Recruitment of Survey Participants. We recruit participants in the following two ways.

(i) *Snowball Approach:* We recruit a list of participants from a set of companies worldwide based on personal contacts. We then adopt a snowballing method [3] to encourage the participants to disseminate our survey to some of their colleagues with similar experiences and willing to participate in our survey. In this process, we confirm 48 participants. However, we allow 44 of them to complete our survey. The remaining four did not satisfy our constraints perfectly.

(ii) *Open Circular:* To find potential participants, we post a description of this study and our research goals in the specialized Facebook groups where professional software developers discuss their programming problems and share software development resources. We also use LinkedIn as a research tool to reach potential participants because it is one of the largest professional social networks in the world. We get 15 participants from this open circular who are willing to participate and satisfy our constraints. Finally, we got nine valid responses from them.

Fig. 6 summarizes the participants' Java experience and professions. Out of the 53 participants, 34 (about 64%) had Java experience five years or less (see Fig. 6a). Developers with comparatively less experience are likely to be more actively engaged in Stack Overflow. However, we recruit 19 participants who had more than five years of Java development experience. The professions of the survey participants were mainly software developers (45.3%), and academic practitioners (e.g., faculty member, student, postdoctoral researcher) (37.7%) (see 6b). However, there were four technical leads and five research engineers. Other details of the participants (e.g., professional software development experience, SO account age, and question answering experience) can be found in our online appendix [2].

3.2.2 Survey Data Analysis. In total, we received 53 valid survey responses. We then analyze the responses with appropriate tools and techniques based on the question types. For multiple-choice questions, we report the percentage of each option selected. To identify the agreement level of each statement, we analyze the Likert-scale ratings. We use Borda count [42] to rank the reproducibility challenges. Furthermore, we extract comments that our survey participants give on the impacts of the reproducibility challenges and tool design requirements.

4 SURVEY RESULTS

In this section, we discuss the survey results and answer our research questions.

4.1 Agreement Analysis to the Catalogue of Reproducibility Challenges (RQ1)

We attempt to find developers' agreement on the challenges (see Table 1) that prevent reproducibility. Such agreement on empirical findings increases confidence in the findings [28].
Approach. We present four SO questions (see Fig. 2, 3, 4 & 5) from the published dataset by Mondal et al. [18]. We first ask the participants to reproduce the question issues using the code segments included with the questions. We then ask either they can reproduce

the issues using the code segments or not. Upon selection, "I cannot reproduce", we present a list of reproducibility challenges that could prevent reproducibility of the issues. We then ask their agreement/disagreement on those reproducibility challenges. Besides, we offer an option to mention if they find additional challenges. On the contrary, upon selection, "I can reproduce", participants are asked to mention their agreement/disagreement with a list of editing actions. However, we offer an option to mention if they perform additional actions.

Our primary focus is to validate the challenge catalog. Thus, we select four questions whose issues could not be reproduced by Mondal et al. [18] and cover all the seven challenges (e.g., Table 1). Here, we ask the following three questions and analyze the participants' agreement in contrast to the reproducibility status (i.e., reproducible/irreproducible) and challenges reported by Mondal et al. [18].

Q₁) Do the participants reproduce the reported issues? (*I can reproduce the issue/I cannot reproduce the issue*)
Q₂) Do the participants agree with the reproducibility challenges? (*Yes/No*)
Q₃) Do the participants find additional challenges reproducing the issues? (*Text*)

Findings. Table 2 shows the summary of the findings. For question 1 (Fig. 2), 94.3% of the participants could not reproduce the issue. Only 5.7% of them report that they could reproduce the issue. We found two reproducibility challenges for this code example that prevent reproducibility. They are - *Class/Interface/Method not found* and *Important part of code missing*. We see that 92% of participants also encounter those two challenges. However, only 8% of them disagree with those challenges. We see a similar agreement/disagreement to the reproducibility status and their associated challenges for the questions 2 (Fig. 3) & 3 (Fig. 4). However, in the case of question 4 (Fig. 5), about 26% of developers can reproduce the reported issue, where the challenge was *outdated code*. The code segment contains classes (e.g., `AudioStream`) that are not being used in JDK-1.8. However, some developers could use previous JDK versions to analyze the code or use alternative classes to reproduce the issue. We ask participants to submit the modified code segments and find that several participants compose new code to reproduce the issue.

In addition to the given challenges, participants report a few additional challenges they encountered as follows.

(1) **Error log/stack trace missing.** Error log or stack trace contains meaningful insights about program failures. Some issues could not be reproduced without such error reports.

(2) **System dependency or environment setup is missing.** Several programming issues are involved with particular Operating Systems (OS) (e.g., Windows), IDE (e.g., Eclipse), and software versions (e.g., Python 3). Some issues could not be reproduced because question submitters do not specify them.

(3) **Sample input-output missing.** Several issues demand sample input-output (i.e., test cases) while reproducing them.

We then analyze the agreement according to the participants' Java experience and profession. Table 3 summarizes the agreement.

Table 2: Developers agreement to the reproducibility status and the challenges that prevent reproducibility

Question No.	Reproducibility Status		Reproducibility Challenges	Reproducibility Challenges	
	Irreproducible	Reproducible		Agree	Disagree
01 (see Fig. 2)	94.3%	5.7%	• Class/Interface/Method not found • Important part of code missing	92% 92%	8% 8%
02 (see Fig. 3)	92.5%	7.5%	• External library not found • Identifier/Object type not found • Too short code snippet	93.9% 89.8% 93.9%	6.1% 10.2% 6.1%
03 (see Fig. 4)	92.5%	7.5%	• Class/Interface/Method not found • Database/File/UI dependency	81.6% 89.8%	18.4% 10.2%
04 (see Fig. 5)	73.6%	26.4%	• Outdated code	79.5%	20.5%
Overall	**88.2%**	**11.8%**	–	**89.1%**	**10.9%**

Table 3
Agreement analysis by experience and profession
(SD: Software Developer, TL: Technical Lead, RE: Research Engineer, AP: Academic Practitioner)

Analysis by Java experience				
Experience	Reproducibility Status		Reproducibility Challenges	
	Irreproducible	Reproducible	Agree	Disagree
≤ 5 years	91.2%	8.8%	85.1%	14.9%
> 5 years	82.9%	17.1%	92.2%	7.8%
Analysis by profession				
Profession	Reproducibility Status		Reproducibility Challenges	
	Irreproducible	Reproducible	Agree	Disagree
SD	86.5%	13.5%	89.8%	10.2%
AP	98.8%	1.2%	85.6%	14.4%
TL & RE	69.5%	30.6%	75.8%	24.2%

We first measure the agreement with reproducibility status and challenges for each of the four example questions and then compute their average. We see that participants with lower experience (e.g., five years or less) agree 8% more on average with the reproducibility status than those having higher experience (e.g., more than five years). Conversely, participants with higher experience agree 7% more with the given reproducibility challenges. Participants with higher experience might have the skill to apply different potential approaches to reproduce the issues. However, low-experienced participants are more enthusiastic about resolving the challenges (e.g., finding external libraries, fixing database dependencies). Analysis by profession shows that technical leads and research engineers agree comparatively less with reproducibility status and challenges. They might not be actively involved in programming and have a hectic task schedule. Thus, they could guess the reproducibility status and the potential challenges. However, our investigation finds that they mainly disagree with the reproducibility status and challenge of question 4 (see Fig. 5), where the challenge was "outdated code".

Summary. About 88% of participants (on average) fail to reproduce the issues using the code segments, and 89% of them encounter the given challenges. Only about 11% of participants (on average) claim that they could reproduce the issues, and thus they do not

agree to the challenges. According to the agreement/disagreement analysis, at least 77% more participants agree to the reproducibility status and challenges on average. Furthermore, our analysis by experience and profession also shows that the lowest agreement level with reproducibility status and challenges is 70% (on average) which is acceptable. Such agreements support and validate the reproducibility status and challenge catalog (Table 1).

4.2 Impacts of Each of the Reproducibility Challenges to Answer Questions (RQ2)

We see the practitioners' agreement to the reproducibility challenge catalog in the previous section (Section 4.1). However, it is important to understand the impact of each challenge to answer a question. Such understanding helps us to determine which challenges need to be addressed early.

Approach. To find the potential impact of the reproducibility challenges, we ask the participants to mark each of the challenges with one of the five categories: (1) not a problem, (2) moderate, (3) severe, (4) blocker, and (5) no opinion. Here, moderate means irritating, but able to answer appropriately. Severe means wasted much time to reproduce the issue, but able to answer, and blocker means could not reproduce the issue and thus could not answer the question. In particular, we ask a question to the participants as follows.

> What are the impacts of the reproducibility challenges to answer a question? (*not a problem/moderate/severe/blocker/no opinion*)

Findings. Fig. 7 shows the summary of how the participants assess the impact of each of the reproducibility challenges. We see that majority of the participants (about 36%) perceive the challenge "Class/Interface/Method not found" as moderate. About 30% of them consider it a blocker. However, only 7.5% of participants consider this challenge not a problem, and the remaining 1.9% of them did not give any opinion. "An important part of code missing" is mostly (about 59% of participants) considered a blocker. We find that "Too short code snippet" is also perceived as a blocker by the majority of the participants (about 47%). About 43% of participants perceive the challenge "Identifier/Object type not found" as severe, whereas 34% of them perceive it as moderate.
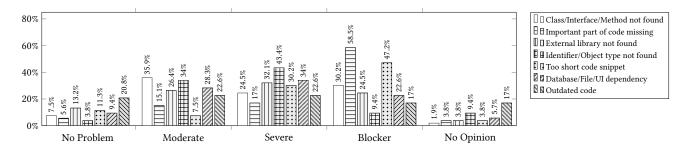
Figure 7: Impacts on the reproducibility challenges answering questions.

We then attempt to find the top challenges in each of the five categories. As Fig. 7 shows, "Class/Interface/Method not found" is the top in the moderate challenge category. The challenge "Identifier/Object type not found" is the top in the severe category, whereas "Important part of code missing" is assessed as the top in blocker category. However, among the reproducibility challenges, "Outdated code" is considered "not a problem" by most participants, or many of them did not give any opinion. To reproduce the issue, developers could migrate the outdated code or use alternative classes/APIs in place of obsolete classes/APIs. Thus, this challenge could be estimated mostly as not a problem among the seven challenges.

In addition to that, we attempt to get more insights into participants' choices of impact. We thus ask their justifications behind the choices. Although it was optional, we receive a few of their comments (see Table 4). For example, one developer states that without the Class/Method definition, it is nearly impossible to reproduce the exact issues.

Summary. Most of the developers could not reproduce the question issues and thus could not answer if code segments miss important statements. Other challenges (e.g., Class/Interface/Method and Identifier/Object type not found) could irritate developers and kill their valuable time. Such factors could also prevent or delay appropriate solutions to the questions. However, developers do not encounter more difficulties in dealing with outdated code.

4.3 Severity Analysis of the Reproducibility Challenges (RQ3)

It is not always possible to fix all the reproducibility challenges due to a strict budget (e.g., limited time). Thus, it is important to prioritize more severe challenges than the others to fix them within a strict budget.

Approach. We ask the following question to participants to find the rank of the reproducibility challenges.

> Which three challenges the participants would like to prioritize above others? (*first choice/second choice/third choice*)

We then use the Borda count as proposed by Yamashita and Moonen [42] to analyze and rank the challenges according to their severity. Borda count is a rank-order aggregation technique for n candidates. The first ranked candidates received n points, the second $n-1$ points, and so on. Since we ask three options to choose, we assign the first option to 3 points, the second to 2 points, and the third to 1 point.
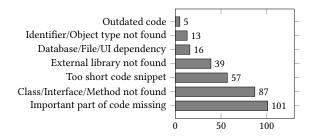

Figure 8: The rank of reproducibility challenges according to their score.

Findings. Fig. 8 shows the challenges with their points. We see that that "important part of code missing" gets the highest (i.e., 101) and "Outdated code" get the lowest (i.e., 5) point. However, we can now prioritize (e.g., rank) the top three challenges according to their severity – (1) important part of code missing, (2) class/interface/method not found, and (3) too short code snippet. This finding of severity analysis is consistent with impact analysis (section 4.2).

Summary. The findings from severity analysis suggest that question submitters should include the part of code (i.e., statements) that could never be guessed, define necessary classes/methods. Moreover, they should not submit too short code segments (e.g., 1/2 lines of code) to receive fast and appropriate solutions.

4.4 Editing Actions to Make the Code Segments Capable to Reproduce the Issues (RQ4)

We could get more insights on how to address the reproducibility challenges by observing the developers' code modification plan.

Approach. We attempt to see the participants' code editing actions to reproduce the question issues. In particular, we want to see how participants mitigate the reproducibility challenges of the code segments. We thus ask participants to submit their modified code segments and manually level the editing actions. We shared the modified code with the level of actions in our online appendix [2].

Findings. Table 5 shows the summary of the code editing actions to make the code segments capable of reproducing the issues. The participants add demo classes and methods, include import statements for the native and external libraries, create objects, declare identifiers and initialize them, and migrate the outdated code to mitigate the reproducibility challenges. In particular, participants first attempt to make the code segments compilable/executable by

Table 4: Practitioners' comments behind the choices of impacts to answer questions

---

"I really think not finding 'Class/Interface/Method' is a blocker problem, because how can I understand a code without their method explained and reproduce. Same case happens in the case of 'Too short code snippet' and 'Important part of code missing'."

"Without Class/Method definition, it's nearly impossible to identify the exact error. Besides, it is helpful, if individuals give some sample input and output examples. Dependency on Database: this is kind of blocker to run the sample code. "

"If the given code part is too small, solution can not be made by depending only few line of code."

"As I already answered most of the time we are irritated to help with question without proper playground. (class not found, Important code missing... etc.)."

Table 5: Editing actions to make the code segments capable of reproducing the question issues.

Question No.	Reproducibility Challenges	Action Detail	No Action
01 (see Fig. 2)	(1) Class/Interface/Method not found, and (2) Important part of code missing	Addition of demo classes and methods (62%), object creation, identifier declaration and initialization (20%), invocation of methods (4%)	36%
02 (see Fig. 3)	(1) External library not found, (2) Identifier/Object type not found, and (3) Too short code snippet	Addition of demo classes and methods (26.5%), inclusion of native and external libraries (14.3%), object creation, identifier declaration and initialization (12.2%), other (8.2%)	61.2%
03 (see Fig. 4)	(1) Class/Interface/Method not found, and (2) Database/File/UI dependency	Addition of demo classes and methods (28%), inclusion of native libraries (12%), other (4%)	68%
04 (see Fig. 5)	Outdated code	Addition of demo classes and methods (20.5%), inclusion of native and external libraries (20.5%), suggest new code (10.3%), other (5.1%)	53.8%

resolving the external dependencies. For example, when methods are invoked without defining them, participants attempt to add their definitions. Participants modify the codes based on the given code segments, hints from question descriptions or guessing. However, we see that 36% – 68% of participants did not modify the code segments. Intuitively, they assume that the code segment is insufficient to reproduce the issues, and thus they did not attempt to modify the code segments. For example, one developer commented, *"I did not try to reproduce it, because it needs much guessing and it may differ much from the original code".*

Summary. Developers' first attempt to make the code segments compilable/executable by resolving all possible dependencies. However, developers do not modify the code segments based on guessing when the code segments are insufficient.

4.5 Tool Support Needs to Promote Reproducibility (RQ5)

Questions whose issues could be reproduced have at least three times higher chance to receive acceptable answers than the questions with irreproducible issues. However, the existing question submission system of SO is not intelligent enough to estimate reproducibility. It does not interact with the question submitters even if the code segments fail to reproduce the question issues. To enhance the possibility of receiving an appropriate answer and save developers' valuable time, we plan to introduce interactive tool support (e.g., browser/IDE plugin) to promote reproducibility.

Approach. We seek participants' recommendations on tool design requirements. Besides, we offer a few tool support options (Table 7) and employ a 5-point Likert scale (i.e., 1–5) to estimate the participants' consent with the tool options. In particular, we ask two questions as follows.

Q$_1$) What kind of tool support do the participants need to assist with the reproducibility challenges? (*Text*)

Q$_2$) How do the participants agree with our tool support options? (*see options from Table 7*)

Findings. Table 6 shows a few valuable recommendations from participants (complete recommendation list can be found in our online appendix [2]). For example, one participant recommended, *"A tool that can sense my written code snippet and see what important part might be missing there".* Besides, Table 7 shows our tool options. We offer six options, such as a tool that warns users about the severe challenges or challenges that could block reproducibility. According to the participants' assessment, three of them are considered very influential (score $\geqslant$ 4.21), and the remaining three are considered influential (3.41 $\leqslant$ score $\leqslant$ 4.20).

Summary. By analyzing the developers' recommendations and assessing the given tool options, we could summarize the primary tool design requirements as follows.

• Interact with question submitters and suggest including the part of code that could never be guessed and mandatory to reproduce the question issues.

Table 6: Practitioners' tool support recommendations to promote reproducibility

"The tool that help me to guess the missing code, library version, missing input data depending on the issue context. It will be great if the tool generates missing information for me depending on the question/issue."
"A software that can ensure the missing cases and force the user to complete them, Auto code generator."
"A tool that helps to guess the missing important code depending on the question context."
"A tool that can sense my written code snippet and see what important part might be missing there."
"It will provide suggestions to reproduce after pasting the code in IDE. "

Table 7: Assessment of the tool support options

Options	Mean Value	Result Interpretation
A tool that suggests users improve the code examples to support reproducibility	4.43	Very Influential
A tool that warns users about reproducibility challenges that are severe or may block the reproducibility	4.25	Very Influential
An IDE (e.g., Eclipse) plugin to support reproducibility	4.25	Very Influential
A browser plugin to support reproducibility	4.02	Influential
A website that guides users to improve reproducibility	3.85	Influential
A static analyzer that examines code segments to find reproducibility challenges	4.15	Influential

• Suggest adding the definition of necessary classes/methods and declaration of identifiers/objects. The absence of such definitions/declarations could prevent reproducibility.

• Warn question submitters not to include a too short code segment. Developers could not guess the actual problem from such a code segment. Sometimes developers guess the problem. However, such guessing often goes wrong, and thus the submitted answer may not solve the actual problem.

• Recommend to include the external libraries when required. It is a time-consuming task to find the external libraries without import statements or any hints about the libraries in the question description.

5 THREATS TO VALIDITY

Threats to internal validity relate to experimental errors and biases [35]. Our key reproducibility challenges were derived from a qualitative study by Mondal et al. [18], which could be a source of subjective bias. However, the challenges were validated by 53 developers with an agreement level of about 90% on average. Moreover, the difference between agreement and disagreement on reproducibility challenges is quite large (about 77%) and statistically significant.

Threats to external validity relate to the generalizability of our findings [35]. We investigate the practitioners' perspective on reproducibility challenges, their impacts and tool design requirements. However, the challenges were derived by analyzing questions related to Java programming problems. Thus, the impacts and tool design requirements can apply to the statically-typed, compiled programming languages such as C++ and C#. Nonetheless, replication of our study using different languages (e.g., dynamically-typed languages like Python) may prove fruitful.

Our survey participants range from novice to experienced (see Fig. 6a) and constitute mainly software developers but also other related professions (see Fig. 6b). Such diversity in the survey participants offers validity and applicability to the survey findings.

However, any individual bias in the survey responses should be mitigated via a large sample of 53 users.

6 CONCLUSIONS

Developers submit thousands of questions to SO daily to resolve their code-level problems. They include an example code segment to support the problem descriptions. Users of SO prefer to reproduce the issues using the code segments and then submit their answers. However, such code segments could not always reproduce the issues due to several unmet challenges that prevent the questions from getting prompt and acceptable solutions. A previous study produces a catalog of challenges (Table 1) that might prevent reproducibility of question issues. However, they are not validated by developers. We survey 53 developers (users of SO) to understand their perspectives on those challenges, impacts of those challenges, how developers address those challenges, and tool design requirements that could mitigate those challenges. Our findings are fourfold. First, about 90% of developers agree to the reproducibility challenge catalog. Second, missing part of code that could never be guessed but required to reproduce question issue mostly prevent questions from receiving answers. Third, developers perform several editing actions (e.g., addition of demo classes/methods) to make the code segments reproduce the issues. However, they do not try to reproduce the issues and submit answers if the code segments are entirely insufficient to guess any actions. Finally, an intelligent tool that identifies the reproducibility challenges according to their severity could help question submitters improve their code segments and reduce the developers' workload who attempt to answer questions.

REFERENCES

[1] B. CD Anda, D. IK Sjøberg, and A. Mockus. 2008. Variability and reproducibility in software engineering: A study of four companies that developed the same system. *TSE* (2008).

[2] Anonymous. 2021. Replication Package. https://bit.ly/3nOS2KM

[3] Tingting Bi, Xin Xia, David Lo, John Grundy, Thomas Zimmermann, and Denae Ford. 2021. Accessibility in Software Practice: A Practitioner's Perspective. *arXiv preprint arXiv:2103.08778* (2021).

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

[5] R. PL Buse and W. R. Weimer. 2009. Learning a metric for code readability. *TSE* (2009).

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

[7] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. 2019. Confusion in code reviews: Reasons, impacts, and coping strategies. In *Proc. SANER*.

[8] M. Erfani J., M. Mirzaaghaei, and A. Mesbah. 2014. Works for me! characterizing non-reproducible bug reports. In *Proc. MSR*.

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

[10] D. Ford, K. Lustig, J. Banks, and C. Parnin. 2018. " We Don't Do That Here" How Collaborative Editing with Mentors Improves Engagement in Social Q&A Communities. In *Proc. CHI*.

[11] Z. Gao, X. Xia, D. Lo, J. Grundy, and Y. F. Li. 2020. Code2Que: A Tool for Improving Question Titles from Mined Code Snippets in Stack Overflow. *arXiv preprint arXiv:2007.10851* (2020).

[12] R. M. Groves, Floyd J. Fowler J., M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau. 2011. *Survey methodology*.

[13] E. Horton and C. Parnin. 2018. Gistable: Evaluating the executability of python code snippets on github. In *Proc. ICSME*.

[14] E. Horton and C. Parnin. 2019. Dockerizeme: Automatic inference of environment dependencies for python code snippets. In *Proc. ICSE*.

[15] A. Joshi, S. Kale, S. Chandel, and D. K. Pal. 2015. Likert scale: Explored and explained. *CJAST* (2015).

[16] B. A. Kitchenham and S. L. Pfleeger. 2008. Personal opinion surveys. In *Guide to advanced empirical software engineering*.

[17] J. C. Lin and K. C. Wu. 2008. Evaluation of software understandability based on fuzzy matrix. In *Proc. FUZZ*.

[18] S. Mondal, M. M. Rahman, and C. K. Roy. 2019. Can issues reported at stack overflow questions be reproduced?: an exploratory study. In *Proc. MSR*.

[19] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. 2016. Automatically discovering, reporting and reproducing android application crashes. In *Proc. ICST*.

[20] D. Mu, A. Cuevas, L. Yang, H. Hu, X. Xing, B. Mao, and G. Wang. 2018. Understanding the reproducibility of crowd-reported security vulnerabilities. In *Proc. USENIX Security*.

[21] Stack Overflow. 2009. Java: Resetting all values in the program. https://stackoverflow.com/questions/798184 Online; Last accessed 10 January 2020.

[22] Stack Overflow. 2009. Use Sounds in java? https://stackoverflow.com/questions/1264770 Online; Last accessed 10 January 2020.

[23] Stack Overflow. 2010. get resource is null. https://stackoverflow.com/questions/2012643 Online; Last accessed 10 January 2020.

[24] Stack Overflow. 2010. Getting Error On Xml resultSet in java. https://stackoverflow.com/questions/2018247 Online; Last accessed 10 January 2020.

[25] Stack Overflow. 2013. Scanner doesn't see after space. https://stackoverflow.com/questions/19509647 Online; Last accessed 10 January 2020.

[26] D. Posnett, A. Hindle, and P. Devanbu. 2011. A simpler model of software readability. In *Proc. MSR*.

[27] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. 2019. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering* (2019).

[28] M. M. Rahman, F. Khomh, and M. Castelluccio. 2020. Why are Some Bugs Non-Reproducible? An Empirical Investigation using Data Fusion. In *Proc. ICSME*.

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

[30] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto. 2016. Improving code readability models with textual features. In *Proc. ICPC*.

[31] J. Singer and N. G. Vinson. 2002. Ethical issues in empirical studies of software engineering. *TSE* (2002).

[32] M. Soltani, A. Panichella, and A. Van D. 2017. A guided genetic algorithm for automated crash reproduction. In *Proc. ICSE*.

[33] M. Tahaei, K. Vaniea, and N. Saphra. 2020. Understanding Privacy-Related Questions on Stack Overflow. In *Proc. CHI*.

[34] V. Terragni, Y. Liu, and S. C. Cheung. 2016. CSNIPPEX: automated synthesis of compilable code snippets from Q&A sites. In *Proc. ISSTA*.

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

[36] C. Treude, O. Barzilay, and M. A. Storey. 2011. How do programmers ask and answer questions on the web?(NIER track). In *Proc. ICSE*.

[37] C. Treude and M. P. Robillard. 2017. Understanding stack overflow code fragments. In *Proc. ICSME*.

[38] A. Trockman, K. Cates, M. Mozina, T. Nguyen, C. Kästner, and B. Vasilescu. 2018. " Automatically assessing code understandability" reanalyzed: combined metrics matter. In *Proc. FSE*.

[39] W. M. Vagias. 2006. Likert-type scale response anchors. *Clemson International Institute for Tourism & Research Development, Department of Parks, Recreation and Tourism Management. Clemson University* (2006).

[40] N. Vincent, I. Johnson, and B. Hecht. 2018. Examining Wikipedia with a broader lens: Quantifying the value of Wikipedia's relationships with other large-scale online communities. In *Proc. CHI*.

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

[42] A. Yamashita and L. Moonen. 2013. Do developers care about code smells? An exploratory survey. In *Proc. WCRE*.

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

[44] 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).