The significance of bug report elements

SOLTANI, M.S.; Hermans, F.F.J.; Bäck, T.H.W.

Empirical Software Engineering (2020) 25:5255–5294
https://doi.org/10.1007/s10664-020-09882-z

The significance of bug report elements

Mozhan Soltani[1] · **Felienne Hermans**[1] · **Thomas Bäck**[1]

Published online: 14 September 2020

Abstract
Open source software projects often use issue repositories, where project contributors submit bug reports. Using these repositories, more bugs in software projects may be identified and fixed. However, the content and therefore quality of bug reports vary. In this study, we aim to understand the significance of different elements in bug reports. We interviewed 35 developers to gain insights into their perceptions on the importance of various contents in bug reports. To assess our findings, we surveyed 305 developers. The results show developers find it highly important that bug reports include crash description, reproducing steps or test cases, and stack traces. Software version, fix suggestions, code snippets, and attached contents have lower importance for software debugging. Furthermore, to evaluate the quality of currently available bug reports, we mined issue repositories of 250 most popular projects on Github. Statistical analysis on the mined issues shows that crash reproducing steps, stack traces, fix suggestions, and user contents, have statistically significant impact on bug resolution times, for ∼70%, ∼76%, ∼55%, and ∼33% of the projects. However, on avarage, over 70% of bug reports lack these elements.

Keywords Software debugging · Mining repositories · Empirical software engineering

1 Introduction

Open source software projects often maintain issue repositories to manage feature requests and bug reports. There are potential advantages to using open issue repositories (Anvik et al. 2005). Contributors of software projects provide their inputs and maintain focused conversations over them. As a result, more bugs in software projects may be identified and fixed (Anvik et al. 2005).

Communicated by: Federica Sarro

✉ Mozhan Soltani
m.soltani@liacs.leidenuniv.nl

Felienne Hermans
f.f.j.hermans@liacs.leidenuniv.nl

Thomas Bäck
t.h.w.baeck@liacs.leidenuniv.nl

[1] Leiden University, Leiden, Netherlands

Bug reports contain various types of information, including: software version, crash description, reproducing steps, reproducing test cases, crash stack traces, and fix suggestions. To make bug reports consistent, often default templates are provided in project repositories, where certain required or at least recommended fields are specified to be filled by the contributors. Yet, the content and therefore quality of bug reports vary (Zimmermann et al. 2010). Potential reasons for this issue include: data loss during a software crash, difficulty to find crash data in log files, and lack of sufficient technical experience (Zimmermann et al. 2010).

If too little data is provided in bug reports, then understanding the problem, and therefore reproducing it is nontrivial and time-consuming. On the other hand, reproducing software crashes is a vital step in software debugging. Developers need to know how to reproduce the crashes to be able to confirm the fixes they deliver. Furthermore, low quality bug reports may demotivate developers and therefore take longer to be processed.

The following are examples of bug reports from various popular projects on Github (2019a; b; c; d; e; f; g; h) . These examples illustrate when a crash stack trace or reproducing test case are missing, developers respond by first asking the bug reporter to provide these elements. Figure 1 shows a bug report (Github 2019a) as well as the responses to the bug report. As Fig. 1a shows, the bug report includes various elements such as actual behavior, reproducing steps, versions of various components, etc. However, the bug report misses a crash stack trace. As Fig. 1b shows, the developers explicitly ask for the crash stack trace. Since after one month, this information is not provided, the bug report is closed.

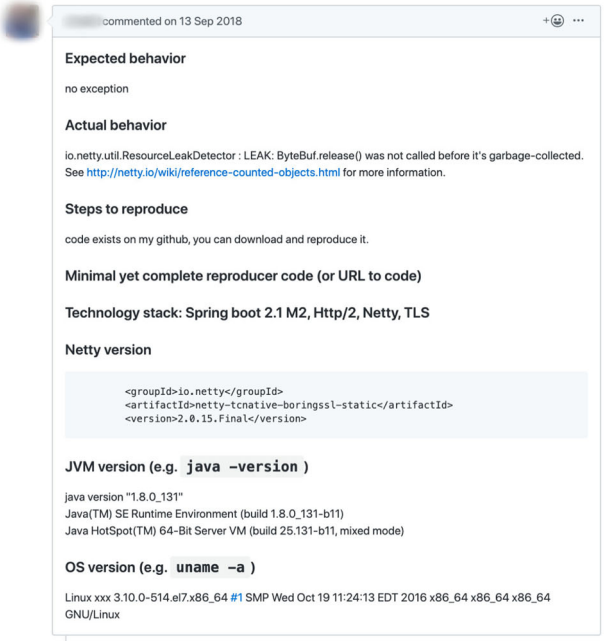
We aim to understand the significance of various information in bug reports for software debugging. To gain an in-depth understanding of developers' perceptions, we interviewed 35 developers. We used Grounded Theory Adolph et al. (2011) and Glaser and Holton (2004) techniques to analyse the interview results. To examine the findings from the interviews, we surveyed 305 developers. Our findings confirm that crash description, crash reproducing steps and test cases, and stack traces are of high importance for developers when debugging. On the other hand, developers find extra information that users may provide such as fix suggestions, code snippets, and links to user content, such as screenshots, of lower importance.

To gain insights on how often important elements are included in bug reports and their impact on bug resolution times, we developed the *IMaChecker* approach. *IMaChecker* receives Github repositories as input, then mines all issues posted in the input repository. Once the issues are downloaded, *IMaChecker* analyses the issues to check whether they are bug reports, and if they contain elements including: crash description, reproducing steps or test cases, stack traces, code snippets, links to user content, or fix suggestions.

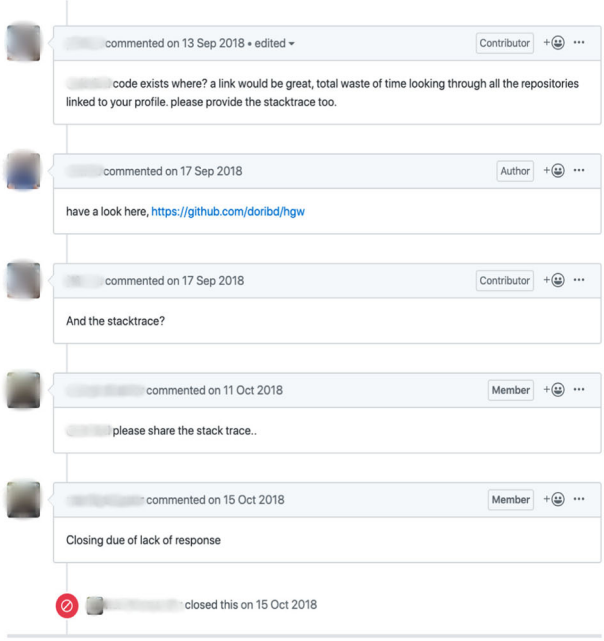
To create a corpus of repositories for evaluation, we first selected five popular languages used in Github according to The State of the Octoverse (2017, 2018) , which are namely: Javascript, Python, Java, PHP, and Ruby. For each language, we selected 50 most popular repositories, resulting in 250 repositories in total, on Github.

To analyse the impact of various elements of bug reports on bug resolution times, we used the Wilcoxon-Mann Whitney test.

To study realistic projects and maintain statistical power, only those projects which provided at least 10 issues for both experimental and control groups, were analysed. Experimental groups contained issues which only included the element of interest in the bug report (e.g., the issue only included stack traces). Control groups contained issues which only included general description of the crash. The results confirm that reproducing steps, stack traces, fix suggestions, and user contents have statistically significant impact on bug



(a) Snapshot of the bug report [1].



(b) Snapshot of the responses to the bug report [1].

Fig. 1 An snapshot of a bug report (Github 2019a) which is missing a crash stack trace, as well as the responses to it

resolution times, for **∼70%, ∼76%, ∼55%, and ∼33%** of the projects, respectively. For code snippets, representative projects were not found.

Furthermore, we used descriptive statistics to report the average percentages of bug reports that include different bug report elements. Despite our findings on important bug report elements and their impact on bug resolution times, on average, over ∼70% of bug reports lack all important elements.

The above results help to raise awareness of the significance of various contents in bug reports for software debugging. Developers can use this information to prepare better templates for bug reports, in which all important elements are explicitly asked for. Furthermore, future work may investigate means to support and enable users to find and provide the information elements.

The contributions of the paper[1] are the following:

1. an extensive report from developer interviews and surveys, in addition to the interview and survey questionnaires,
2. *IMaChecker* as an open source tool, written in Python, which can be used to mine and analyse issues from Github repositories, and
3. a reproducible package which contains the data set of all mined issues from 250 most popular Github repositories, together with the R scripts used to analyse the mined data.

The remainder of this paper is organized as following: Section 2 presents the research methodology. Section 3 presents the IMaChecker approach. Section 4 presents the results. Section 5 provides discussion on the findings of the paper. Section 6 provides related work. Finally, Section 7 concludes the paper.

2 Research Methodology

The overarching goal of this study is to identify the significance of elements of bug reports for software debugging. Therefore we define the following research questions:

– **RQ_1:** What types of information do developers perceive as important in bug reports?

Motivation: The quality of bug reports varies depending on the kinds of information which are included in them. The study by Zimmermann et al. (2010) shows developers and users of Apache, Eclipse, and Mozilla find reproduction steps and crash stack traces to be the most useful elements in bug reports. However, there is little knowledge about the other elements in bug reports and the extent to which they are perceived as important for software debugging. We raise $\mathbf{RQ}_1$ to broaden our perspective and gain a holistic understanding about the extent to which different bug report elements are of importance for software debugging in developers' perception.

Data collection and analysis: To answer $\mathbf{RQ}_1$ we aim to combine interviewing developers with surveying them. By conducting interviews, we intend to gain a preliminary understanding of developers' views on bug reports and the role that each bug report element plays in the process of software debugging. We use thematic analysis to analyse the interview data. Using the information from the interviews, we devise a survey study where we examine and quantify the results from the interviews. We use descriptive statistics to measure the percentages of participants who consider a bug

[1] The interview and survey questions, as well as the dataset package are available via the following DOI: 10.5281/zenodo.3666763

report element as highly important, moderately important, slightly important, or not important for software debugging.

– **RQ_2:** Do the important elements in bug reports impact bug resolution times?

  Motivation: While with RQ_1 we identify the extent to which different bug report elements are important in developers' perception, it would still be unclear in real-world practice, what impact these elements may have on bug resolution times. Therefore, we raise RQ_2 to understand the effect of different bug report elements on the time it takes to resolve bug reports. By raising RQ_2, we intend to evaluate and correspond the extent the bug report elements that software developers perceive important for software debugging, actually impact the time it takes to resolve bug reports.

  Data collection and analysis: To answer RQ_2, we use Github APIs to mine bug report repositories from Github. Once we obtain the bug reports from Github, we use the IMaChecker technique (presented in Section 3) to parse the bug reports statically. Once the static analysis is done, we then use statistical tests to measure the impact of various bug report elements on bug resolution times.

– **RQ_3:** How often do bug reports contain the important elements?

  Motivation: With RQ_1 and RQ_2, we gain an understanding about the extent to which different bug report elements are important for bug resolution. However, it would still be unclear how often these important elements are actually provided in bug reports. For example, as the study by Zimmermann et al. (2010) shows, elements such as crash stack traces are difficult to provide.

  Data collection and analysis: To answer RQ_3, we use the results from the static analysis which is performed by IMaChecker on the mined bug reports. As a result of this analysis, different elements of bug reports are identified. Therefore, we use descriptive statistics to report how often various elements appear in bug reports.

By combining qualitative and quantitative research methods, we use a mixed-method research approach (Creswell and Creswell 2017) to answer the research questions. In what follows, we further present the research techniques we used.

2.1 Interviews

To answer RQ_1, we followed a qualitative research method (Creswell and Creswell 2017). We interviewed 35 developers in order to gain an understanding of their debugging techniques and the kind of information they find important to receive in bug reports. In what follows, we present the interview protocol, the participants, and data analysis technique we used for the interviews.

2.1.1 Protocol

We conducted semi-structured interviews (Hove and Anda 2005), in which we combined broad and open-ended questions[2] with specific questions. In this way, we let participants freely respond and explore relevant topics, while we made sure the intended topics were also explored by asking specific questions. As suggested by Barriball and While (1994) and Jacob and Furgerson (2012), we conducted four pilot interviews before we performed the main interviews. As a result, we received feedback on the general flow of the questions from

[2]The interview questions are provided in the reproduction package, via DOI: 10.5281/zenodo.3666763

two of the pilot interviews. According to this feedback, we should have noted the role the participants play in their organization. Therefore, we added two questions in the interview instrument where we specifically ask about the role of the participant and we ask if the participant can briefly explain what this role entails.

We let the participants know in advance that we intend to use the data anonymously. Prior to the interviews we got permission from the participants to record the interviews. Furthermore, 15 out of 35 interviews were conducted through online calls because the developers were not available in person. Each interview took between 20 minutes to 60 minutes.

2.1.2 Participants

We intended to form a diverse group of participants. Thus, using our social contacts, we reached out to developers who work in the following areas: e-commerce development, ERP application development, automotive industry, artificial intelligence, embedded programming, and database administrating. We sent personalized emails to 50 developers who worked in these industries. 40 people with background in e-commerce development, ERP application development, and automotive industry agreed to participate in this study. After 35 interviews we reached theoretical saturation (Glaser and Holton 2004). Figure 2 shows the years of professional experience of the interview participants. The participants had at least five and at most 25 years of professional experience as a developer.

2.1.3 Data Analysis

After the interviews, we manually transcribed the recorded interviews. To analyse the collected data, we used thematic analysis Gibbs (2007) and Braun and Clarke (2006) to identify emerging categories in the transcripts. Thematic analysis is a technique that is used when analysing textual data. Using this technique, the first author read the transcripts intensively. The first author then used open and axial coding techniques (Moghaddam 2006) to tag the pieces of text which would relate to $\mathbf{RQ}_1$. After identifying the tags, the first author reviewed them and grouped them together to form more generic themes. Ultimately, the identified themes addressed two main categories: the debugging techniques developers used,

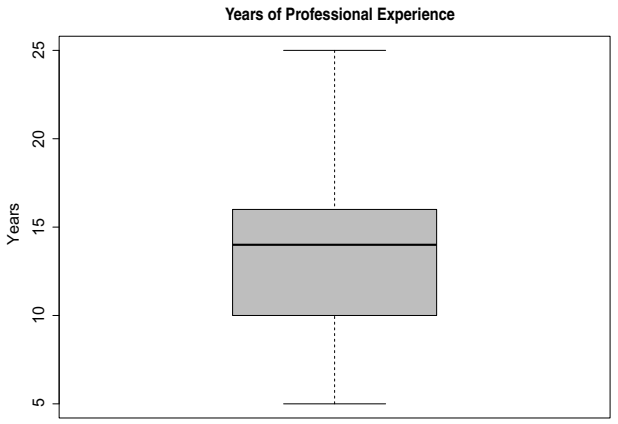


Fig. 2 The years of professional experience of the interview participants

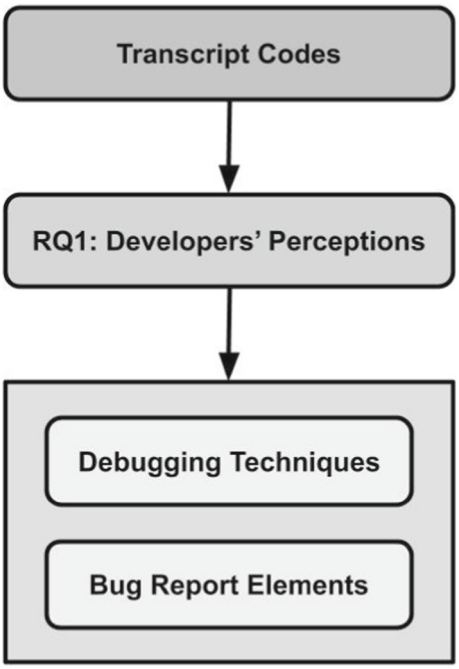


Fig. 3 The identified themes after analysing the interview transcripts

and the kind of information in bug reports they considered important for software debugging. Figure 3 is a visual representation of the main themes that were identified throughout this process.

2.2 Surveying Developers

To generalize the findings from the interviews, and measure the prevalence of the debugging practices and developers' perceptions on the importance of different bug report elements for software debugging, we surveyed 305 developers. In what follows, we describe the survey protocol, survey participants, and our data analysis approach.

2.2.1 Protocol

To construct the survey[3], we used guidelines from Fink (2003), De Vaus and de Vaus (2013), Pfleeger and Kitchenham (2001, 2002). We used closed questions to make the survey more compelling for the participants to fill in. To avoid forcing the participants to choose an option, for each closed question, there was an option where the participants could write their responses. We provided a brief overview of the purpose of the survey in the introduction. We let participants know we would use the data anonymously.

Before sending out the survey, we used pilot studies with four participants who were professional developers. We asked the participants to fill in the survey, and provide us with their feedback about the structure and questions of the survey. One feedback we received was about the length of the introduction at the beginning of the survey. The participant

[3]The survey questions are provided in the reproduction package, via DOI: 10.5281/zenodo.3666763

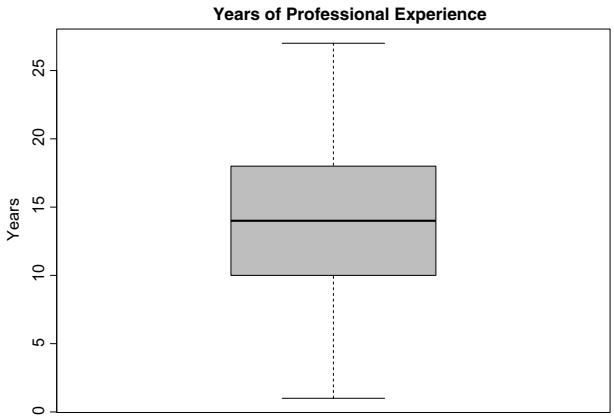


Fig. 4 The figure presents the years of professional experience of the survey participants

mentioned that the introduction could be shortened for more readability. In addition, another feedback was about asking the participants if they wish to receive the results after the survey is done. This is why we added one last question at the end of survey where the participants can leave their contact information if they wish to receive the results. We discarded the results of the pilot studies from the main results in this paper.

2.2.2 Participants

To find participants for the survey, we searched for trending developers[4]. Trending developers on Github are developers who maintain the most popular projects on Github. In addition, we searched for active developers from 85 popular software projects on Github. The main rationale behind this approach for selecting the participants is that we intended to involve participants who are selected from a pool of experienced developers. We considered experienced developers to be those who have been maintaining projects on Github. Typically, these developers have more than at least one year of experience in software development. Nevertheless, we did not consider a strict threshold for the minimum number of years of experience in software development by the respondents. From each project we selected three to four active developers. This way we reached out to 317 people. We sent personalized emails to these developers, and briefly explained the purpose of the study to them. We received 222 responses. In addition, we used the snowballing technique (Myers and Newman 2007) to collect more participants. After the participants responded to the survey, we asked them if they could introduce us to colleagues who would be interested to participate in the study. In our request, we mentioned that we intended to get in touch with experienced colleagues who know the project well and have been contributing to the project for at least the past year. We relied on the judgment of the respondents to connect us with the colleagues who would fulfill this criterion. We sent personalized emails to 105 developers, and we received 83 responses. In total, we received 305 responses for the survey. Figure 4 shows the years of professional experience of the survey participants.

[4]Through https://github.com/trending/developers

2.2.3 Data Analysis

To analyse the results of the survey, we used descriptive statistics to report the findings from the closed questions. Therefore, for each bug report element, we simply measure the percentages of participants who perceive the element as highly important, moderately important, slightly important, or not important. Furthermore, we count the number of participants who are project manager, software developer, software tester, software maintainer, scrum master, or those who indicate any other type of role they play. We also count the number of years of professional experience the participants indicate to have. For the questions which let the participants write an answer in text, we use thematic textual analysis to identify emerging categories from the written texts.

2.3 Mining Github Issues

To answer $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, we mined and analysed issues from 250 projects on Github. To do so, we developed the Issue Miner and Checker (IMaChecker) approach. IMaChecker mines the issues of the received repositories, and further checks them to detect whether stack traces, reproducing steps, fix suggestions, code snippets, and user content are provided in the issues. In Section 4 we will further describe the IMaChecker approach.

To select the projects, we first identified the five most popular programming languages used in Github. According to The State of the Octoverse (2017, 2018), the languages are: Javascript, Python, Java, PhP, and Ruby. Next, based on the measures of popularity that Borgens et al. identify (Borges et al. 2015), for each language, we selected 50 projects, 250 projects in total, that have the most number of stars and forks. Table 4 (in Appendix A) presents an overview of the projects, the number of stars, forks, contributors, as well as the year in which the first commits were provided in the project[5].

2.3.1 Analysis of the Mined Issues

To measure the impact of various elements of bug reports on bug resolution times, we use the Wilcoxon-Mann Whitney statistical test. This is a non-parametric test that is used to analyse the impact of an independent variable that is at least ordinal. When it is not possible to make assumptions about whether the data is normally distributed or not, Wilcoxon-Mann Whitney is an alternative approach that can be used instead of techniques such as the independent samples t-test. Since in this case the dependent variable is resolution time, we only consider closed issues where the reported bug is fixed. The null hypotheses in these experiments are the following:

- $\mathbf{H}_0 1$**:** the time it takes to close a bug report which only includes a problem description and crash stack trace is the same as the time it takes to close a bug report that only includes a problem description.
- $\mathbf{H}_0 2$**:** the time it takes to close a bug report which only includes a problem description and reproduction steps is the same as the time it takes to close a bug report that only includes a problem description.
- $\mathbf{H}_0 3$**:** the time it takes to close a bug report which only includes a problem description and fix suggestion is the same as the time it takes to close a bug report that only includes a problem description.

[5]The results were collected on 2019-05-15.

- **$H_0$4:** the time it takes to close a bug report which only includes a problem description and user content is the same as the time it takes to close a bug report that only includes a problem description.
- **$H_0$5:** the time it takes to close a bug report which only includes a problem description and code snippet is the same as the time it takes to close a bug report that only includes a problem description.

We use experimental and control groups. In experimental groups, only those issues are present which only include one of the bug report elements e.g., stack traces, depending on the element under analysis. Control groups contain those issues in which none of the bug report elements are present. To analyse realistic projects and maintain statistical power, we make sure that the sample sizes are at least 10, i.e., at least 10 issues are analysed in each group. Furthermore, the test does not assume that the samples are normally distributed. We consider α=0.05 for Type I errors to assess the significance of the results.

We use the Vargha-Delaney $\hat{A}_{12}$ statistic (Vargha and Delaney 2000) to assess the effect sizes. Vargha-Delaney $\hat{A}_{12}$ is also a non-parametric approach for comparing performances of two independent groups. The outcome of this test is a value between 0 and 1. Therefore, if the outcome of Vargha-Delaney $\hat{A}_{12}$ is 0.5, the two groups perform the same. On the other hand, if the result of Vargha-Delaney $\hat{A}_{12}$ is less than 0.5, the first group performs worse, while if the outcome is larger than 0.5, the first group perform better than the second group. The closer $\hat{A}_{12}$ is to 0.5, the smaller the difference between the two groups is. Furthermore, when the first group performs better than the second group, $\hat{A}_{12}$ is considered small when it is between 0.6 and 0.7, while it is considered medium, when it is between 0.7 and 0.8. If $\hat{A}_{12}$ is larger than 0.8, then it is considered large. We note that these thresholds are arbitrary to some extent. Using Vargha-Delaney $\hat{A}_{12}$, we report effect magnitudes which indicate the following effect sizes: negligible, small, medium, and large.

3 The IMaChecker Approach

To mine and analyse the issues, we developed the Issue Miner and Checker (IMaChecker) in Python 3. This approach has been tested on a Linux kernel version 4.15, as well as a MacOS 10.14 machine.

Figure 5 presents an overview of the approach. *IMaChecker* receives the list of Github Repositories as input. Next, IMaChecker downloads all issues posted to the repository, using

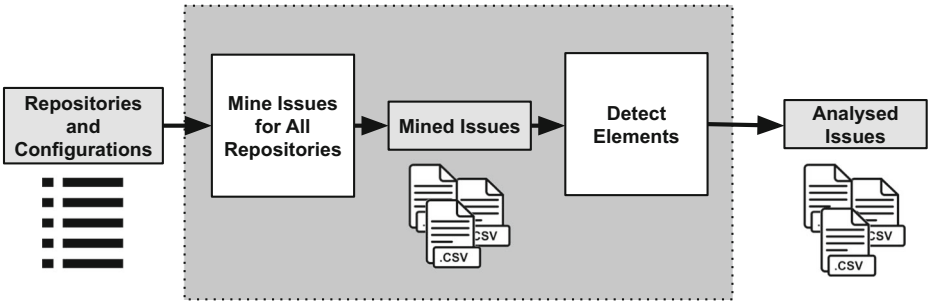


Fig. 5 The figure presents an overview of the *IMaChecker* Approach

Table 1 The strings and regular expressions we used to parse reproduction steps, fix suggestions, user contents, and code snippets in bug reports

Element	Strings or regular expressions
Reproduction Step	"reproducing steps"
	"steps to reproduce"
	"reproduce"
	"reproducible test case"
	"reproducible"
	"to reproduce"
	"minimal reproduction"
Fix Suggestion	"fix suggestion"
	"suggestion to fix"
	"suggestions to fix"
	"suggest"
	"suggestions"
User Content	https://user-images.githubusercontent.com/[Sa-z0-9A-Z]+.[a-zA-z]
Code Snippet	[\ w+\ s]+```` \\ r \\ n

the Github API (developer.github 2015). After the issues of all projects are downloaded, the user can use the APIs that *IMaChecker* provides to analyse the downloaded issues.

IMaChecker uses regular expressions to detect issues that are originally labeled as bugs. Often various terms (e.g. "crash") are used to mark an issue as a bug in issue repositories. Therefore, it is possible to feed *IMaChecker* with specific terms of interest to detect originally labeled bugs.

IMaChecker uses specific strings and regular expressions to detect whether the issues include stack traces, reproducing steps, fix suggestions, code snippets, and links to user contents. To identify the strings and design the regular expressions, we studied 255 bug reports which were randomly selected from the projects presented in Table 4 (in Appendix A). After we reached the saturation point and did not find any new keys in the context of the bug reports, we collected a pool of strings which were commonly used to refer to different bug report elements. Table 2 shows these strings and regular expressions Table 1.

Since each programming language uses a specific format to generate stack traces, *IMaChecker* uses five different regular expressions that are adjusted to the five different stack trace formats in Javascript, Python, Java, PhP, and Ruby. Table 2 shows examples of stack traces for different languages as well as the regular expressions used to detect them.

If *IMaChecker* detects a stack trace in the issue, the exception type of the stack trace is recorded as well. This can be used when one wishes to report frequency of various exception types. In addition, if *IMaChecker* detects crash reproducing steps or stack traces, or fix suggestions, then it automatically marks the issue as a bug. This can be useful as not always the issues are labeled in a Github Repository.

Table 2 Examples of stack traces in different languages as well as the regular expressions used to detect them

Language	Example	Regular Expression
Javascript	at split (angular.js:27114) at updateClasses (angular.js:27043) ..., from Github (2019i)	[\s]+at[\s]+[\w+.]+[\s]+\([/*\w+]+.js:[0-9]+:*[0-9]*\)\\r\\ n
Python	Traceback (most recent call last): File "facedetect.py", line 251, in <module> main_loop() ..., from Github (2019j)	Traceback\s\Smost\srecent\scall\s last\S:|File[\s].+,[\s]+line[\s] +[0-9]+,[\s]+in[\s]+.+\\ r\\ n
Java	at android.view.ViewGroup. dispatch-Draw(ViewGroup.java:3554) at android.view.View.updateDisplay ListIfDirty(View.java:15237) ..., from Github (2019k)	[\s]+at[\s]+[\w+.\S]+\(\w+.java :[0-9]+\)
PHP	#0/Applications/MAMP/htdocs/ learning/laravel/larabootstrap5/ vendor/laravel/framework/src/ Illuminate/Foundation/Bootstrap/ HandleExceptions.php(118): ..., from Github (2019l)	\#[0-9]+\s+[\w+\S]+.php \([0-9]+\):
Ruby	/home/navin/.rvm/gems/ruby-2.2.1/gems/sprockets-3.4.0/lib/sprockets/sass_processor.rb:278:in sprockets_context ..., from Github (2019m)	[\w+\S]+.rb:[0-9]+:in\s+

Furthermore, Fig. 6 shows an example[6] of a bug report from the AngularJS project. As the example shows, this bug report contains a description of a memory allocation problem together with a snapshot that is included as a .png file. When IMaChecker parses the bug report content, it detects the user content is provided through the .png file.

To evaluate the precision of the IMaChecker approach, we randomly selected 100 bug reports from the projects in Table 4 (in Appendix A). We manually analyzed the bug reports and made an account of the elements included in them. We then ran IMaChecker in order to detect the bug report elements automatically. The precision was around 92%. This was because there were bug reports in which reproduction steps or stack traces were provided through user contents (e.g., through links to external pages). Therefore, it was not possible for the IMaChecker approach to detect these elements by parsing the texts.

4 Results

We used a mixed-method research approach to discover the significance of bug report elements in software debugging. To answer the research questions, we combined interviewing developers with surveying them. In addition, we mined 250 issue repositories and used descriptive statistics as well as statistical tests on the mined issues. In this section, we present the results and thereby answer $\mathbf{RQ}_1$, $\mathbf{RQ}_2$, and $\mathbf{RQ}_3$.

[6]This bug report can be found via: https://github.com/angular/angular.js/issues/16853

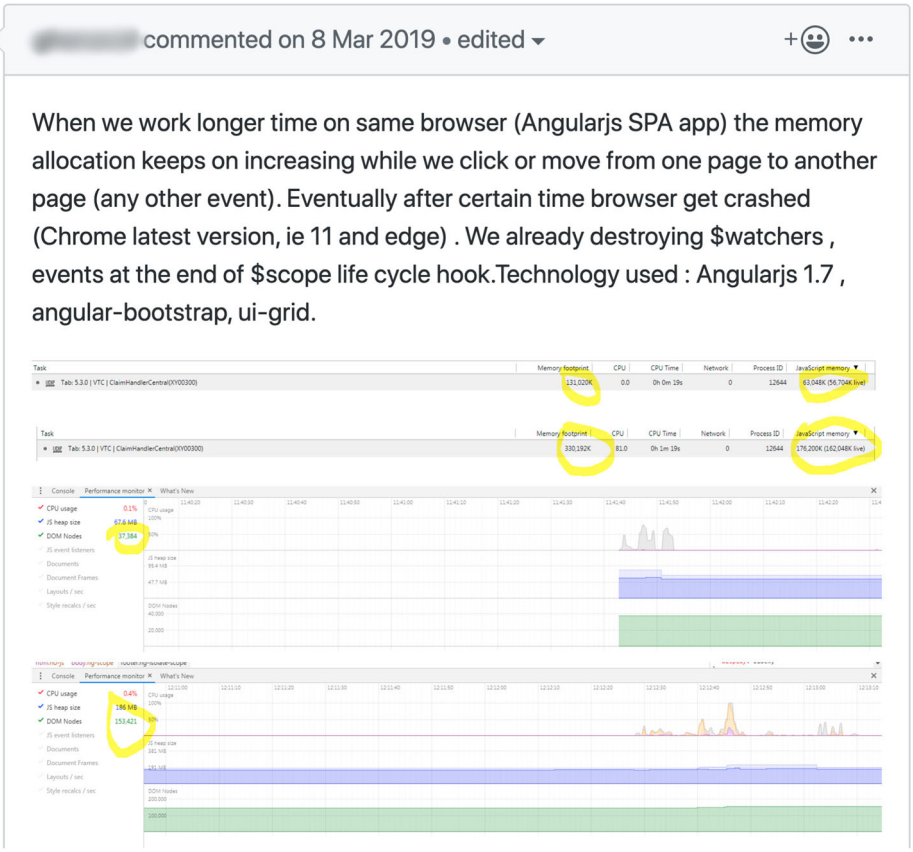


Fig. 6 An example of a bug report from the AngularJS project

4.1 RQ_1. What Types of Information do Developers Perceive as Important in Bug Reports?

During the interviews, in order to get a broad understanding of the debugging process the developers have, we asked the participants to describe the debugging approach they take typically. In this regard, we gained the following insights. When picking up a bug report to process, developers first try to reproduce the problem on their side. This step is important because not only developers confirm there is a problem to fix, but also by being able to reproduce the problem, developers can evaluate the fixes they provide.

After reproducing the problem, developers need to assess the state of the programs at different stages of the executions. To do so, the interview participants often prefer using *printf*s for debugging. When a crash is complex, then 45% of the interview participants indicated they would use a debugger to further analyse the execution scenarios. In addition, all participants indicated that especially when they face a new error they have not seen before, they typically google the error message. Often it is the case that on platforms such

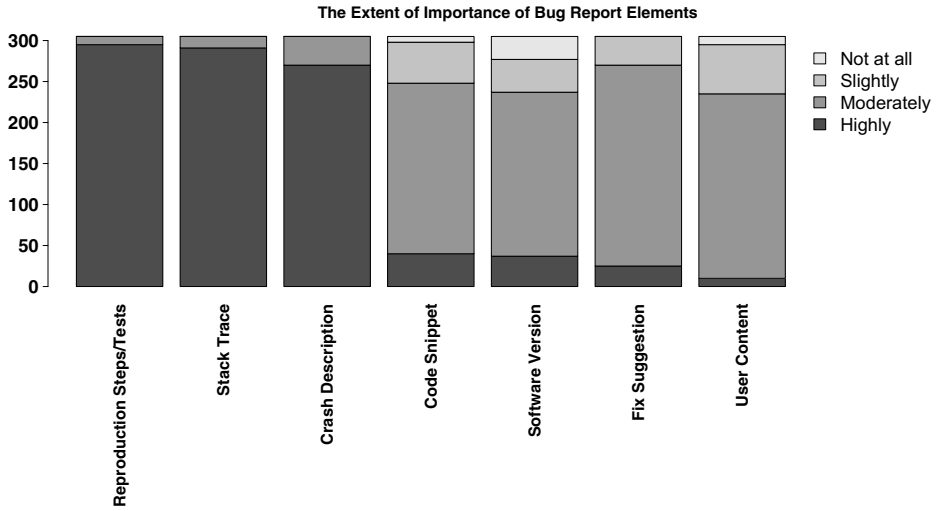


Fig. 7 Developers' perception on the importance of various data for bug resolution time

as *stackoverflow*[7], someone else has posted a similar problem, which provides the participants an opportunity to get further insights. Otherwise, they may open a new issue on those platforms, share their problem, and ask the community to look into the questions.

To answer RQ_1, we derived 7 categories from the interview results which indicate the information elements that developers perceive as important, which they prefer to be included in bug reports: crash description, software version, reproduction steps, stack traces, code snippets, user content and fix suggestions. To quantify these results and gain insights into the extent to which these elements are of importance for debugging, we surveyed more developers.

According to the survey results, 38% of the respondents mentioned that they use default templates for bug reports in their projects. As Fig. 7 presents, 96% of the participants find reproduction steps or test cases of high importance while 4% of them believe reproducing steps or tests are moderately important. 95% of the participants find crash stack traces of high importance while 5% of them find crash stack traces of moderate importance.

In addition, around 89% of the participants find crash description of high importance, while 11% of them believe crash descriptions are of average importance. Around 12% of the participants find software version of high importance, while 66% of them believe software versions are of average importance.

Around 14% of the participants find code snippets of high importance, while 68% of them believe code snippets are of average importance. 16% of the participants find code snippets of slight importance. 2% of the participants find code snippets of no importance for software debugging. In this regard, a participant mentioned: "I prefer to receive them in a pull request not in a bug report."

13% of the participants find software versions of slight importance. 9% of the participants do not find software version important for software debugging. One of the participants indicated: "Often the version is understood from the context of the bug report. For example, certain features are only available in our latest release."

[7] https://stackoverflow.com/

Around 8% of the participants find fix suggestions of high importance, while around 81% of them believe fix sugestions are of average importance. 11% of the participants believe fix suggestions are of little importance.

Around 3% of the participants find user contents of high importance, while 74% of them believe user contents are of average importance. 19% of the participants find user contents of slight importance. 3% of the participants find user contents of no importance for software debugging. In this regard, a participant mentioned: "User content could be anything. They are supplementary."

4.2 RQ_2. Do the Important Elements in Bug Reports Impact Bug Resolution Times?

Table 3 presents the results of Wilcoxon-Mann Whitney and Vargha Delaney $\hat{A}_{12}$ statistical analysis on four elements of bug reports, namely: stack traces, crash reproducing steps or test cases, fix suggestions, and user contents.

Since we compare resolution times, we only consider closed issues where the reported bug is fixed. To maintain statistical power, we made sure that in each project, there are at least 10 issues which have none of the comparison elements in the description (control group), and there are at least 10 issues which have only the comparison factor (e.g., stack traces) in the description (experimental group). If a project does not provide such groups, we excluded it from the analysis.

To analyse the impact of stack traces, we found 139 projects, which provide the control and experimental groups. In 106 projects out of 139 projects (∼76%) statistically significant results show that including stack traces impacts the bug resolution times. For 33 projects (∼24%) no conclusion was drawn.

To analyse the impact of reproducing steps or test cases, we found 142 projects, which provide the control and experimental groups. In 100 projects out of 142 projects (∼70%) statistically significant results show that including reproducing steps impacts the bug resolution times. For 42 projects (∼30%) no conclusion was drawn.

To analyse the impact of fix suggestions, we found 148 projects, which provide the control and experimental groups. In 81 projects out of 148 projects (∼55%) statistically significant results show that including fix suggestions impacts the bug resolution times. For 67 projects (∼45%) no conclusion was drawn.

To analyse the impact of user contents, we found 33 projects, which provide the control and experimental groups. In 11 projects out of 33 projects (∼33%) statistically significant results show that including user contents impacts the bug resolution times. For 22 projects (∼67%) no conclusion was drawn.

4.3 RQ_3. How Often do Bug Reports Contain the Important Elements?

To identify how often various bug report elements are included in bug reports, we used *IMaChecker*[8] to mine and analyse issue repositories from 250 Github projects. In total, 835381 issues were mined, out of which 89761 issues (∼11%) were open while 745620 issues (∼89%) were closed. 114053 bug reports (∼29.64%) were originally labeled as bugs in bug repositories while 219803 bug reports (∼70.36%) were automatically detected.

According to the results, for 228 projects, crash reproducing steps and stack traces were detected. For 244 projects fix suggestions were detected. For 226 projects user contents

[8]The mining was done on 2019-05-13.

Table 3 The table shows the results from the Wilcoxon-Mann Whitney, and Vargha Delaney $\hat{A}_{12}$ statistical analysis on four elements of bug reports, namely: Stack Traces, crash Reproducing Steps, Fix Suggestions, and User Contents

	Stack Trace		Reproducing Step		Fix Suggestion		User Content	
Repository	p	v-mag.	p	v-mag.	p	v-mag.	p	v-mag.
30-seconds/30- seconds-of-code	–	–	0.094	m	0.561	n	–	–
activeadmin/ activeadmin	0	l	0	l	0	m	–	–
adobe/brackets	–	–	0	m	0	s	–	–
angular/angular.js	–	–	0	m	0	s	–	–
ansible/ansible	0	m	0	n	0.001	n	0.015	m
apache/incubator-dubbo	0	m	0.005	s	0.125	s	–	–
apache/incubator -echarts	–	–	0	m	0.485	n	0.745	n
apache/incubator-zipkin	0.09	s	0.146	s	0.026	m	0.499	n
atech/postal	0.517	s	–	–	–	–	–	–
atom/atom	0	l	0	m	0	s	–	–
axios/axios	0.048	s	0.088	s	0.714	n	–	–
babel/babel	0	s	0.02	n	0	s	0.002	m
bazelbuild/bazel	0.002	s	0.002	n	0.059	n	–	–
bcit-ci/CodeIgni ter	–	–	0.701	n	0.281	n	–	–
BetterErrors/better_errors	0.209	s	–	–	–	–	–	–
briannesbitt/Carbon	–	–	0.025	m	0.503	s	–	–
bumptech/glide	0.02	n	0.17	n	0.046	s	–	–
CachetHQ/Cachet	0	s	0	s	0.065	s	0	l
cakephp/cakephp	0.003	m	0.047	s	0.389	n	–	–
capistrano/capistrano	0	l	0	l	0.001	m	–	–
carrierwaveuploader/carrierwave	0	l	0	l	0	l	–	–
celery/celery	0	m	0	s	0.001	s	–	–
certbot/certbot	0	s	0.026	s	0.046	n	–	–
chartjs/Chart.js	–	–	0	s	0.001	s	0.172	n
chrisbanes/PhotoView	0	l	–	–	–	–	–	–
composer/composer	0.002	m	0	s	0	s	–	–
deeplearning4j/deeplearning4j	0	s	0.412	n	0.979	n	0.507	n
deployphp/deployer	–	–	0	s	0.357	n	–	–
diaspora/diaspora	0.081	n	0.014	n	0.009	s	–	–
dingo/api	0	l	0.009	m	0	l	–	–
docker/compose	0.005	n	0	s	0.697	n	–	–
Dogfalo/materialize	0	l	0	l	0	l	–	–
elastic/elasticsearch	0.786	n	0	n	0	s	–	–
elastic/logstash	0.001	n	0	s	0.031	n	–	–
encode/django-rest-framework	0	m	0	m	0	s	–	–
explosion/spaCy	0.002	s	0.025	n	0.945	n	–	–
expressjs/express	0.006	s	0.003	s	0.036	s	–	–
facebook/create-react-app	0	m	0	s	0.615	n	–	–
facebook/fresco	0	m	0	m	0.025	s	0.015	m

Table 3 (continued)

Repository	Stack Trace		Reproducing Step		Fix Suggestion		User Content	
	p	v-mag.	p	v-mag.	p	v-mag.	p	v-mag.
facebook/react	0	l	0	l	0	s	–	–
facebook/react-native	0	m	0.969	n	0.12	n	0.051	s
facebook/stetho	0.77	n	–	–	–	–	–	–
fastlane/fastlane	0	s	0.898	n	0	s	0.478	n
fluent/fluentd	0.001	s	0.04	s	0.193	s	–	–
FortAwesome/Font-Awesome	–	–	0	m	0	s	0.546	n
freeCodeCamp/devdocs	0.523	n	0.124	s	0.006	l	–	–
freeCodeCamp/freeCodeCamp	–	–	0	l	0.072	n	0.7	n
FriendsOfPHP/PHP-CS-Fixer	–	–	0.681	n	0.117	s	–	–
gatsbyjs/gatsby	0.007	s	0	s	0.649	n	0.218	s
getgrav/grav	0.057	s	0.602	n	0.178	n	–	–
getredash/redash	0	m	0.037	n	0.021	m	0.158	s
getsentry/sentry	0.094	s	0.968	n	0.01	s	–	–
github/linguist	0.071	m	–	–	0.327	s	–	–
gollum/gollum	0	m	0.084	s	0.059	s	–	–
google/ExoPlayer	0	m	0	s	0	s	0.029	m
GoogleChrome/puppeteer	0	m	0.001	s	0.394	n	–	–
greenrobot/greenDAO	0.405	n	–	–	–	–	–	–
gulpjs/gulp	0	l	0	l	0	l	–	–
guzzle/guzzle	0.005	l	0.092	s	0.318	s	–	–
h5bp/html5-boilerplate	–	–	0.083	m	0.028	s	–	–
hakimel/reveal.js	–	–	0.115	s	0.407	n	–	–
hashicorp/vagrant	0	l	0	s	0	s	–	–
HelloZeroNet/ZeroNet	0.146	s	0.313	n	0.312	s	–	–
home-assistant/home-assistant	0.007	s	0.925	n	0.595	n	–	–
Homebrew/brew	0	l	0	m	0	m	–	–
huge-success/sanic	0.054	s	–	–	0.752	n	–	–
huginn/huginn	0.046	m	–	–	0.567	n	–	–
imathis/octopress	0	m	0.735	n	0.096	s	–	–
ipython/ipython	0	m	0	s	0.043	n	–	–
jakubroztocil/httpie	0.02	s	–	–	–	–	–	–
javan/whenever	0	l	–	–	0.001	l	–	–
jordansissel/fpm	0.116	s	–	–	–	–	–	–
jquery/jquery	–	–	0	l	0	l	–	–
kaminari/kaminari	0	l	0.108	s	0.236	s	–	–
kennethreitz/requests	0	l	0	l	0	l	–	–
keras-team/keras	0.048	s	0.02	s	0.005	m	–	–
Konloch/bytecode-viewer	0.865	n	–	–	–	–	–	–
laravel/framework	0	l	0	l	0	l	–	–
localstack/localstack	0.251	n	0.529	n	–	–	–	–

Table 3 (continued)

	Stack Trace		Reproducing Step		Fix Suggestion		User Content	
lodash/lodash	0.058	s	0.021	s	0.01	s	–	–
magento/magento2	0	m	0	s	0	m	–	–
matomo-org/matomo	0.699	n	0.244	n	0	s	0.28	n
meteor/meteor	0	l	0	s	0	s	–	–
Microsoft/vscode	0	s	0	n	0.889	n	0.007	n
middleman/middleman	0	l	0.001	m	0.09	s	–	–
mikepenz/MaterialDrawer	0.541	n	0.299	n	0.054	s	–	–
mitmproxy/mitmproxy	0	m	0.002	s	0.563	n	–	–
mockery/mockery	–	–	0.003	l	0.026	m	–	–
moment/moment	–	–	0	m	0.005	s	–	–
monicahq/monica	0.595	n	0.192	s	–	–	0.011	s
mrdoob/three.js	0.002	m	0.002	s	0.005	n	0.696	n
mui-org/material-ui	0	l	0	m	0	m	0	m
mybatis/mybatis-3	0.615	n	0.196	n	0.876	n	–	–
NationalSecurityAgency/ghidra	–	–	0.731	n	–	–	0.819	n
netty/netty	0.002	s	0	s	0	m	–	–
nextcloud/server	0.458	n	0.549	n	0.033	n	0.505	n
nicolargo/glances	0.333	n	–	–	0.998	n	–	–
nodejs/node	0	m	0	s	0.049	n	–	–
nostra13/Android-Universal-Image-Loader	0.001	m	–	–	0.005	m	–	–
octobercms/october	0	l	0	s	0	m	0.057	s
omniauth/omniauth	0.01	l	–	–	0.406	s	–	–
pallets/flask	0	l	0.025	m	0.021	m	–	–
pandas-dev/pandas	0	s	0	s	0.512	n	–	–
parcel-bundler/parcel	0	m	0	s	0.029	s	0.896	n
phalcon/cphalcon	0	s	0	s	0.528	n	–	–
phanan/koel	–	–	0.038	s	0.014	m	–	–
PhilJay/MPAndroidChart	0.365	n	0.226	s	0.24	n	–	–
plataformatec/devise	0	l	0	m	0	l	–	–
plataformatec/simple_form	0	l	0.018	m	0	l	–	–
prettier/prettier	0	l	0.005	s	0.674	n	0.271	s
pypa/pipenv	0	l	0	l	0	m	–	–
rapid7/metasploit-framework	0	s	0	m	0.428	n	–	–
react-native-community/lottie-react-native	–	–	0.06	m	–	–	–	–
ReactiveX/RxJava	0.637	n	0	m	0.042	s	–	–
ReactTraining/react-router	0	l	0	l	0	l	–	–
realm/realm-java	0	s	0	s	0.01	s	–	–
reduxjs/redux	–	–	0.002	l	0.203	s	–	–
resque/resque	0	l	–	–	0.007	m	–	–
roots/sage	–	–	0	l	0.01	m	–	–
rubocop-hq/rubocop	0	m	0.447	n	0.23	n	–	–
ruby-grape/grape	0	s	–	–	0.525	n	–	–

Table 3 (continued)

	Stack Trace		Reproducing Step		Fix Suggestion		User Content	
scikit-learn/scikit-learn	0	s	0	m	0.2	n	–	–
scrapy/scrapy	0	m	0.02	m	0.013	s	–	–
sebastianbergmann/phpunit	0.167	n	0.491	n	0.305	n	–	–
Seldaek/monolog	0.002	l	–	–	0.004	l	–	–
Semantic-Org/Semantic-UI	0.577	n	0.99	n	0.149	n	0.008	m
serverless/serverless	0.001	s	0.346	n	0.089	n	0.635	n
sferik/rails_admin	0.005	s	0.457	n	0.954	n	–	–
Shopify/liquid	0.001	l	–	–	–	–	–	–
signalapp/Signal-Android	0	l	0	l	0	l	–	–
sinatra/sinatra	0.001	m	0.189	s	0.407	n	–	–
skylot/jadx	0.191	s	–	–	–	–	–	–
slimphp/Slim	0.992	n	0.478	n	0.12	s	–	–
socketio/socket.io	–	–	0.001	s	0.018	s	–	–
spring-projects/spring-boot	0	n	0	s	0.069	n	–	–
spring-projects/spring-framework	0	n	0	s	0	s	–	–
sqlmapproject/sqlmap	0.359	n	0.054	n	0.031	s	–	–
square/okhttp	0	m	0	m	0.002	s	–	–
square/retrofit	0.001	l	0.131	m	0.057	m	–	–
StevenBlack/hosts	0.93	n	–	–	0.467	s	–	–
storybooks/storybook	0.037	s	0	s	0.507	n	0.67	n
stympy/faker	0.002	l	–	–	–	–	–	–
symfony/symfony	0	m	0	s	0.587	n	0.034	s
teamcapybara/capybara	0	l	0	l	0	l	–	–
Tencent/tinker	0	l	–	–	–	–	–	–
tensorflow/models	0	m	0	l	0.001	m	–	–
the-control-group/voyager	–	–	0	s	0.372	n	0.652	n
thepracticaldev/dev.to	–	–	0	l	0.763	n	0.049	s
thoughtbot/bourbon	–	–	0	l	0.001	l	–	–
thoughtbot/factory_bot	0.01	m	0.325	s	0.317	s	–	–
thoughtbot/paperclip	0	l	0	l	0.005	s	–	–
tmuxinator/tmuxinator	0.006	m	0.827	n	–	–	–	–
tootsuite/mastodon	0	s	0	m	0.622	n	0.902	n
trailofbits/algo	–	–	0.007	s	0.873	n	–	–
TryGhost/Ghost	0	l	0	s	0.001	s	0.515	n
twbs/bootstrap	0.009	m	0	s	0	m	–	–
twbs/bootstrap-sass	0.626	n	0.909	n	0.067	s	–	–
vuejs/vue	0	l	0	l	0	m	–	–
webpack/webpack	0	l	0	l	0	s	–	–
wix/react-native-navigation	0.707	n	0.825	n	1	n	–	–
yarnpkg/yarn	0.016	s	0.029	n	0.429	n	–	–
yiisoft/yii2	0	s	0	s	0	n	–	–

Table 3 (continued)

	Stack Trace		Reproducing Step		Fix Suggestion		User Content	
ytdl-org/youtube-dl	0	m	0	m	0	l	–	–
zeit/next.js	0	m	0	s	0.067	n	–	–
zxing/zxing	0.001	l	0	l	0	l	–	–

p indicates the p values from the Wilcoxon test. **v-mag.** indicates the Vargha Delaney measures of magnitude, which show the effect sizes. **l**,**m**,**s**, and **n**, indicate large, medium, small, and negligible effect sizes, respectively. **-** indicates that control or experimental groups were not found for the comparison factor

were detected. For 178 projects code snippets were identified. Finally, for 34 projects no bugs were originally labeled while IMaChecker detected bugs.

For *kilimchoi/engineering-blogs*, *doctrine/inflector*, and *doctrine/lexer* repositories no issues were originally or automatically marked as bugs. These repositories have 66, 27, and 2 issues, respectively. For these repositories, no reproducing steps, stack traces, fix suggestions, code snippets, or user contents were detected. For more detailed results, please see Table 5 in Appendix B.

In addition, Fig. 8 presents the average percentages of different bug report elements. According to Fig. 8, on average, ∼27.16% of the bug reports included stack traces, ∼27.07% of the bug reports included reproducing steps, and ∼20.59% of the bug reports included fix suggestions. In addition, on average, ∼14.23% of the bug reports included user contents, and ∼1.06% of the bug reports included code snippets.

5 Discussion

In this paper, we aim to identify the contents in bug reports that are of importance for debugging. Therefore, we sought for developers' perceptions in this regard, we analysed

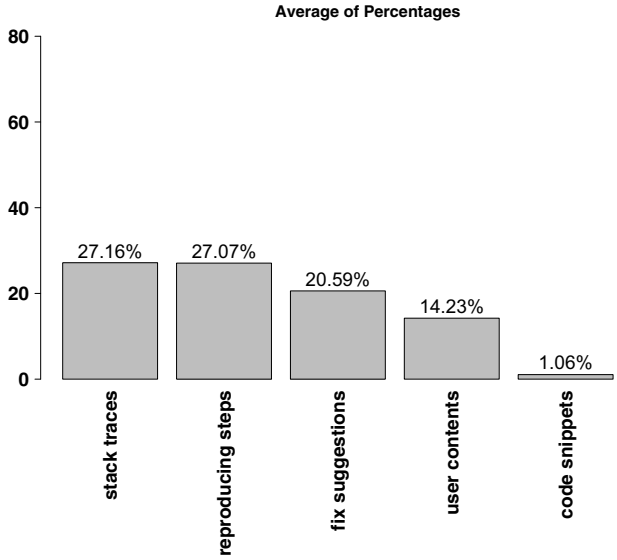


Fig. 8 Average percentages of various elements of bug reports

whether any of the bug report elements impact bug resolution times, and we measured how often various information elements are included in bug reports.

Our results show that certain elements, namely: crash description, reproducing steps, and stack traces are of high importance for debugging in developers' perceptions. According to the statistical analysis, reproducing steps, stack traces, fix suggestions, and user contents have statistically significant impacts on bug resolution times. Despite the above findings, as Fig. 8 shows, on average, over ∼70% of the bug reports lack these elements. These findings indicate that in order to support developers for more efficient software debugging, it is important to include the aforementioned elements when reporting bugs. Furthermore, it is also important to understand why over ∼70% of the bug reports lack the important elements despite the impact they can have on efficient debugging. Understanding the underlying reasons in this regard may help seek approaches to ensure quality of bug reports is maintained high. In what follows we further discuss the findings.

5.1 Bug Report Templates and User Support

In order to keep the issues consistent, and make sure certain elements are provided in bug reports, repositories often provide templates for reporting issues. The specified elements in such templates vary. While these templates often specify reproducing steps, or fix suggestions as fields to be filled by the users, stack traces, user contents or code snippets are not mentioned in the templates. Therefore, it is up to the issue reporter to provide them.

Our results show each of those elements, particularly stack traces, impact the bug resolution times. Therefore, to help keep the structure of issues consistent, and make sure important elements of bug reports are asked for, it is important to provide complete and well-structured bug report templates. The results presented in this paper help increase awareness in this regard.

On the other hand, as Zimmermann et al. (2010) report, it may not be possible for users to provide certain information in their bug reports while at the same time it is important to do so. It is simply because important information are not always easy to be found. For example, stack traces are often hidden in log files, and therefore, it is not easy to find them, even if the issue templates ask for them. Therefore, future work may investigate means to support users and enable them to provide important information in bug reports.

5.2 Representative Samples

When analysing the impact of various bug report elements, many projects were excluded from the analysis because they did not offer representative samples for experiment and control groups. This is why it was not possible to analyse the impact of code snippets on bug resolution times.

The automated mechanism in *IMaChecker* helps increase the number of bug reports, thereby the sample sizes for experimental groups. *IMaChecker* detects whether an issue is a potential bug if a certain element such as stack trace or fix suggestion is included in the reported issue.

However, if an issue does not include any of the elements, the only way to identify whether it is a bug report would be to check the labels put on the issue. At the same time, many of the bug reports were not originally labeled as bugs. Therefore, they could not be used in the control groups. As a result, many projects were excluded from the analysis.

This observation highlights the importance of properly documenting the bug reports. The *IMaChecker* approach provides a more accurate overview of the issues if bug reports are properly marked by developers.

5.3 Internal Validity of the Experiments

Internal validity of a study refers to how well the findings of the study explain a claim about a cause and effect. In the context of our study, threats to internal validity refer to alternative reasons why a bug report is closed more quickly than others.

In some cases, bug reports are created, however they either have no content or very minimal amount of information. We have observed that these kinds of bug reports are typically very quickly closed because there is not much that can be done for them. When developers close such bug reports, often they ask the contributors who opened the bug reports to provide further information. Furthermore, sometimes bug reports are re-opened. One possible explanation is that the issue, which was addressed previously, resurfaces, either for the same contributor who previously opened the issue or someone else.

In our experiments, IMaChecker automatically checks the contents in experimental groups and control groups before they are included in the statistical tests. Therefore, the bug reports used in these experiments are never entirely empty. However, it could be that they are closed because they included too little information. In addition, in these experiments, we do not check whether an issue is re-opened later on. This is mainly due to the fact that the information that can be retrieved through the Github API does not include sufficient details with regards to whether the issue was re-opened or not.

5.4 Construct Validity

Before conducting the interviews and survey, we performed pilot studies with four professional developers. We received feedback from the pilot studies, incorporated the feedback before performing the main interviews and survey, and discarded the results of the pilot studies from the main results reported in this paper. We note that while we intended to discover which bug report elements are important specifically for efficient handling of the reported bugs, the questions we asked in the interviews and survey did not explicitly clarify what we mean by elements being important for software debugging. This can be a potential threat to the construct validity of the study in that the survey respondents may have had their own interpretation of what being important is, based on which they may have responded to the questions.

5.5 Generalizability of Results

As Basili et al. (1999) discuss, carrying out empirical work in software engineering is complex and time consuming. They argue that one reason for such complexity is that there are a large number of context variables. Therefore, creating a cohesive understanding of the experimental results requires effort.

We selected participants from three different industries, e-commerce, ERP, and automotive. In addition, the survey participants were either trending developers on Github or selected from over 85 distinct popular software projects. The professional experience of these participants ranged from one year to 27 years. While we intended to involve experienced developers in the survey, we did not ensure if the developers have experience in developing closed source projects or not.

To make a corpus of open-source projects, we selected 250 projects from Github. Github is a popular platform where over 2 million organizations and 96 million repositories are hosted to which over 31 million developers contribute, according to The State of the Octoverse (2019). To select the open source projects, first we chose five popular programming

languages, and then we used common measures of popularity, i.e., number of stars and forks, to identify the projects. Furthermore, we used statistical tests to analyse the results.

However, we can not claim that the findings are transferable to closed-source projects. Communication with users and debugging practices differ in closed-source projects. Future work may investigate closed-source projects as well as expert developers in the field, and compare the results with the findings reported in this paper.

5.6 Automated Crash Reproduction

Depending on the available information and complexity of the reported crash, reproducing the crash may be a complex and time consuming task for developers. Researchers have proposed several approaches to automated crash reproduction. The state of the art techniques are: STAR (Chen and Kim 2015), EVOCRASH (Soltani et al. 2018), and JCHARMING (Nayrolles et al. 2015).

Each of the proposed approaches have certain advantages and limitations, which are to some extent reported in Soltani et al. (2018). Upon further advances in this direction, automated crash reproduction may compensate for lack of crash reproducing steps in bug reports.

5.7 What Do User Contents Provide?

The results show that user contents have statistically significant impact on bug resolution times for ∼33% of the projects. User contents are provided through a link in the bug reports. However, their contents vary. In our manual analysis, we found out that the links may refer to long stack traces that the users preferred to provide separately from the main bug report. It is also possible for user contents to address fix suggestions or UI features. Future work may investigate the kinds of data provided through user contents and their frequencies. Such investigation helps analyse the impact of user contents more accurately.

5.8 Ethics Approval

Typically, prior to conducting human research, researchers obtain ethics approval from the organization they are a member of. While in some countries and academic organizations, it is mandatory to obtain ethics approval, in the country where the authors conducted the reported research, obtaining ethics approval is an optional task.

Even though the authors did not obtain ethics approval, they did follow certain standard guidelines when interviewing and surveying participants which are reported in Section 2. In particular, the authors made sure to inform the participants about the goals and context of the studies prior to the interviews and surveys, and how the data will be used while keeping the data anonymous. In addition, prior to the interviews, the authors asked the participants for permission for recording the interviews.

6 Related Work

To understand what makes a good bug report, Zimmermann et el. (2010) conducted a survey among developers and users of Apache, Eclipse, and Mozilla. They found out that across all three projects, crash reproducing steps, and stack traces, are most useful. At the same time these types of information are most difficult for users to provide. Their results show, to a large extent, lack of tool support causes this mismatch. For example, while stack traces

are hidden in log files, experienced users of Eclipse know that Error logs exists. Therefore, experienced users can provide stack traces while for other users it is difficult to do so (Zimmermann et al. 2010).

In addition, Zimmermann et al. (2010) asked developers to rate 289 bug reports, that were selected randomly, from very poor to very good, using a five-point Likert scale (Likert 1932). They use the rated bug reports to train the CUEZILLA approach they propose. CUEZILLA measures the quality of bug reports, and recommends which elements should be added to improve the quality of bug reports.

This paper builds on the work by Zimmermann et al. (2010) in that we interviewed and surveyed developers to understand their perceptions on the importance of different bug report elements. However, while Zimmermann et al. (2010) surveyed the developers and users of Apache, Eclipse, and Mozilla, our approach to finding interview and survey participants were different. We first found participants from ERP, E-commerce, and automotive industries to execute the interviews. We used the insights from the interviews to construct a survey study where we contacted active developers from 85 different trending projects on Github. Furthermore, while CUEZILLA uses developers' ratings to measure the quality of bug reports, IMaChecker takes a different approach for analyzing the bug reports. IMaChecker statically parses the bug reports from 250 projects (developed in five different languages) to identify which elements are present in the bug reports, and using this information, IMaChecker applies statistical tests to identify the impact of the bug report elements on bug resolution times. Our findings with regards to the impact of bug report elements on bug resolution times are aligned with the findings reported by Zimmermann et al. (2010) in that the results from interviews, surveys, and statistical tests show crash reproduction steps and stack traces are most useful for processing bug reports. Furthermore, despite the indicated importance, our results show that the majority of times, these elements are not included in bug reports.

Schroter et al. (2010) conducted an empirical study with the Eclipse project to understand the extent to which stack traces are useful when debugging. Their findings show that the average lifetimes of bug reports which include stack traces are significantly lower than of other bugs. Furthermore, their findings show up to 60% of bug reports which included stack traces involved changes to one of the stack frames.

In this paper, we expand the findings reported by Schroter et al. (2010) in that we study bug reports from 250 projects to assess the impact of several different bug report elements, including crash stack traces. Our results on the importance of crash stack traces for bug resolution times are aligned with the findings reported by Schroter et al. (2010).

With regard to characterizing bug report quality, Hooimeijer and Weimer (2007) provide a descriptive model based on a statistical analysis of 27000 publicly available bug reports for the Mozilla Firefox project. The proposed model predicts whether a reported bug is fixed within a given amount of time.

With regards to estimating the time it take to fix a bug report, Zeng and Rine (2004) present a non-parametric approach based on using dissimilarity matrix and self-organizing neural networks. They used NASA's KC1 data set to evaluate their approach. The results indicated that their clustering approach performs well when applied on a family of products such as software projects in product lines. However, the defect fix estimation performed poorly when applied on software projects from different environments. Moreover, Weiss et al. (2007) propose an approach that automatically predicts the time it takes to fix a bug. Given a new reported issue, their technique finds similar older issues and uses their resolution time for prediction. They evaluated their approach using effort data from JBoss project. For bug reports, their technique is off by one hour.

In this paper, rather than providing a prediction model for estimating the time it takes to fix a bug, we use statistical tests to show how different bug report elements impact the time it takes to close bug reports. Furthermore, rather than looking into a single case study, we studied bug reports from 250 open source projects from Github.

7 Conclusions

Software projects often have open issue repositories. Bug reports that are submitted to issue repositories have varying contents. Therefore it is important to gain understanding about the significance of different elements in bug reports.

To understand the extent to which developers percieve various types of information important, we interviewed 35 developers. To asses the findings, we further surveyed 305 developers. The results show crash description, reproducing steps, and stack traces are of high importance in developers' perceptions.

To identify how often the important information elements are provided in bug reports, and what their impact is on bug resolution times, we developed *IMaChecker* to mine and analyse issues from Github repositories. Our statistical analysis, on issues from 250 projects on Github, confirms that crash reproducing steps, stack traces, fix suggestions and user contents have statistically significant impact on bug resolution times. However, on average, over ∼70% of the bug reports of a given repository lack these elements. Future work may investigate means to support users and developers for providing high quality bug reports.

Acknowledgments We cordially thank all interview and survey participants for their invaluable contributions.

Appendix A

Table 4 shows the corpus of 250 open source projects we selected from Github.

Table 4 This table shows the repositories we use in the evaluation

Repository	Since	Stars	Language	Forks	Contributors
30-seconds/30-seconds-of-code	2017	43.1k	Javascript	4.7k	164
achael/eht-imaging	2016	4.6k	Python	414	9
activeadmin/activeadmin	2010	8.4k	Ruby	2.9k	569
adam-p/markdown-here	2012	37.3k	Javascript	6.3k	12
adobe/brackets	2011	29.7k	Javascript	6k	355
ageitgey/face_recognition	2017	23.7k	Python	6.2k	23
airbnb/lottie-android	2016	25.3k	Java	3.9k	71

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
androidannotations/androidannotations	2010	10.7k	Java	2.4k	56
angular/angular.js	2010	59.5k	Javascript	28.9k	1595
ansible/ansible	2012	37.1k	Python	15.1k	4372
apache/incubator-dubbo	2012	25.9k	Java	17.2k	198
apache/incubator-echarts	2013	33.6k	Javascript	9.8k	71
apache/incubator-zipkin	2015	10.9k	Java	1.9k	78
atech/postal	2017	8.9k	Ruby	522	14
atom/atom	2011	48.5k	Javascript	11.4k	431
axios/axios	2014	58.2k	Javascript	4.5k	164
aymericdamien/TensorFlow-Examples	2015	30.9k	Python	11.7k	54
babel/babel	2012	32.8k	Javascript	3.4k	724
barryvdh/laravel-debugbar	2013	9.3k	PHP	905	95
barryvdh/laravel-ide-helper	2013	8.2k	PHP	782	107
bazelbuild/bazel	2015	11.9k	Java	1.9k	441
bcit-ci/CodeIgniter	2006	17.2k	PHP	7.6k	441
BetterErrors/better_errors	2012	6.5k	Ruby	430	75
binux/pyspider	2014	13k	Python	3.2k	51
bobthecow/psysh	2012	7.4	PHP	216	48
briannesbitt/Carbon	2012	12.4k	PHP	1k	197
bumptech/glide	2013	26k	Java	9k	96
CachetHQ/Cachet	2014	9.6k	PHP	1.1k	161
cakephp/cakephp	2005	7.8k	PHP	3.4k	523
capistrano/capistrano	2013	11k	Ruby	1.7k	215
carrierwaveuploader/carrierwave	2008	8.3k	Ruby	1.4k	326
celery/celery	2009	12.3k	Python	3.2k	714
certbot/certbot	2012	25k	Python	2.5k	352
chartjs/Chart.js	2013	43k	Javascript	9.5k	298
chrisbanes/PhotoView	2012	15.2k	Java	3.5k	34
CocoaPods/CocoaPods	2011	11.6k	Ruby	2k	266
composer/composer	2011	19.6k	PHP	5.4k	729
daimajia/AndroidSwipeLayout	2014	11.1k	Java	2.6k	16
daimajia/AndroidViewAnimations	2014	10.5k	Java	2.2k	17
deeplearning4j/deeplearning4j	2013	10.7k	Java	4.6k	250
deployphp/deployer	2013	6.7k	PHP	977	174
diaspora/diaspora	2010	12.2k	Ruby	2.9k	342
dingo/api	2014	8.3k	PHP	1.1k	96
docker/compose	2013	16k	Python	2.4k	299
doctrine/inflector	2009	7k	PHP	90	55
doctrine/instantiator	2014	6.8k	PHP	42	22
doctrine/lexer	2013	6.8k	PHP	29	16
Dogfalo/materialize	2014	35.6k	Javascript	4.7k	252
donnemartin/system-design-primer	2017	62.9k	Python	9.2k	65

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
egulias/EmailValidator	2013	6.7k	PHP	91	37
elastic/elasticsearch	2010	40.3k	Java	13.4k	1205
elastic/logstash	2009	10.2k	Ruby	2.7k	398
encode/django-rest-framework	2010	14k	Python	4.1k	851
EnterpriseQualityCoding/FizzBuzzEnterpriseEdition	2012	10.8k	Java	505	30
erusev/parsedown	2013	10.8k	PHP	881	39
eugenp/tutorials	2013	14k	Java	20.4k	500
explosion/spaCy	2014	13.2k	Python	2.2k	333
expressjs/express	2009	43.4k	Javascript	7.4k	220
facebook/create-react-app	2016	66.5k	Javascript	14.8k	672
facebook/fresco	2015	15.5k	Java	3.6k	152
facebook/react	2013	127k	Javascript	23.2k	1296
facebook/react-native	2015	76.2k	Javascript	17k	1947
facebook/stetho	2015	11k	Java	1k	49
facebookresearch/Detectron	2018	20.3k	Python	4.3k	27
faif/python-patterns	2012	20.4k	Python	4.4k	86
fastlane/fastlane	2014	25.4k	Ruby	3.8k	961
filp/whoops	2013	10k	PHP	523	99
fluent/fluentd	2011	7.8k	Ruby	913	169
FortAwesome/Font-Awesome	2018	59.5k	Javascript	10k	5
freeCodeCamp/devdocs	2013	20.5k	Ruby	1.3k	93
freeCodeCamp/freeCodeCamp	2013	302k	Javascript	21.6k	3532
FriendsOfPHP/Goutte	2010	7.2k	PHP	871	66
FriendsOfPHP/PHP-CS-Fixer	2012	7.5k	PHP	203k	1k
gatsbyjs/gatsby	2015	33.9k	Javascript	4.8k	1954
getgrav/grav	2014	10.8k	PHP	1k	148
getredash/redash	2013	12.5k	Python	2k	247
getsentry/sentry	2008	20.7k	Python	2.3k	383
github/linguist	2011	6.7k	Ruby	2.4k	748
gollum/gollum	2010	9.9k	Ruby	1.4k	144
google-research/bert	2018	14.8k	Python	3.4k	26
google/ExoPlayer	2014	12.9k	Java	3.9k	135
google/gson	2008	15.5k	Java	3.1k	93
google/guava	2011	31.1k	Java	7k	185
google/python-fire	2017	14k	Python	818	28
GoogleChrome/puppeteer	2017	48.2k	Javascript	4.2k	208
greenrobot/greenDAO	2011	11.2k	Java	2.7k	6
gulpjs/gulp	2013	31.1k	Javascript	4.4k	216
guzzle/guzzle	2011	16.6k	PHP	1.9k	294
h5bp/html5-boilerplate	2010	42.6k	Javascript	10.1k	231
hakimel/reveal.js	2011	45.8k	Javascript	13.2k	245
hashicorp/vagrant	2010	18.4k	Ruby	3.7k	884

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
hdodenhof/CircleImageView	2014	11.7k	Java	2.6k	12
HelloZeroNet/ZeroNet	2015	13.7k	Python	1.7k	101
home-assistant/home-assistant	2013	23.5k	Python	6.8k	1441
Homebrew/brew	2009	17.6k	Ruby	3.9k	669
Homebrew/homebrew-cask	2012	15.2k	Ruby	7.2k	5214
huge-success/sanic	2016	12k	Python	1.1k	206
huginn/huginn	2013	21.3k	Ruby	2.3k	171
iluwatar/java-design-patterns	2014	46.8k	Java	15.1k	145
imathis/octopress	2009	9.5k	Ruby	2.9k	111
impress/impress.js	2011	34.7k	Javascript	6.8k	63
Intervention/image	2013	9.2k	PHP	1k	71
ipython/ipython	2008	13.5k	Python	3.8k	593
JakeWharton/butterknife	2013	23.7k	Java	4.5k	83
jakubroztocil/httpie	2012	41.1k	Python	2.6k	74
javan/whenever	2009	7.9k	Ruby	685	82
jekyll/jekyll	2008	37.7k	Ruby	8.2k	852
jfeinstein10/SlidingMenu	2012	11.1k	Java	5.3k	21
jordansissel/fpm	2011	9.1k	Ruby	915	234
josephmisiti/awesome-machine-learning	2014	39.8k	Python	9.7k	371
jquery/jquery	2006	51.4k	Javascript	18k	275
juliangarnier/anime	2016	30.7	Javascript	2.2k	27
kaminari/kaminari	2011	7.4k	Ruby	958	133
kennethreitz/requests	2011	38.6k	Python	6.9k	533
keon/algorithms	2016	14.9k	Python	2.7k	105
keras-team/keras	2015	41.1k	Python	15.3k	795
kilimchoi/engineering-blogs	2015	15.2k	Ruby	1.7k	303
Konloch/bytecode-viewer	2014	10.1k	Java	637	15
laravel/framework	2013	17.2k	PHP	6.4k	1944
lgvalle/Material-Animations	2015	12.7k	Java	2.5k	9
LMAX-Exchange/disruptor	2011	10.3k	Java	2.6k	31
localstack/localstack	2016	16.7k	Python	1.1k	157
lodash/lodash	2009	38.7k	Javascript	22.5k	280
loopj/android-async-http	2011	10.4k	Java	4.2k	75
Maatwebsite/Laravel-Excel	2013	6.9k	PHP	1.1k	95
magento/magento2	2011	7.3k	PHP	6.3k	1129
mame/quine-relay	2013	7.8k	Ruby	383	12
matomo-org/matomo	2007	11.1k	PHP	1.7k	224
matterport/Mask_RCNN	2017	11.9k	Python	5.1k	40
meteor/meteor	2011	41k	Javascript	5k	402
Microsoft/vscode	2015	73.9k	Javascript	10k	871
middleman/middleman	2009	6.4k	Ruby	694	182

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
mikepenz/MaterialDrawer	2014	10.3k	Java	2k	87
minimaxir/big-list-of-naughty-strings	2015	32.3k	Python	1.3k	56
mitmproxy/mitmproxy	2010	14.9k	Python	1.9k	253
mockery/mockery	2009	7.7k	PHP	356	139
moment/moment	2011	40.9k	Javascript	6.1k	492
monicahq/monica	2017	7.1k	PHP	838	158
mperham/sidekiq	2012	9.6k	Ruby	1.6k	397
mrdoob/three.js	2010	50.7k	Javascript	19k	1077
mui-org/material-ui	2014	46.1k	Javascript	9.9k	1229
mybatis/mybatis-3	2010	10.6k	Java	6.6k	121
NARKOZ/hacker-scripts	2015	34.9k	Javascript	5.9k	41
NationalSecurityAgency/ghidra	2019	15.3k	Java	1.8k	40
netty/netty	2008	18.8k	Java	8.4k	373
nextcloud/server	2010	7.4k	PHP	1.3k	601
nicolargo/glances	2011	13.3k	Python	906	92
nikic/PHP-Parser	2011	10.4k	PHP	614	82
nodejs/node	2009	60.3k	Javascript	13.4k	2444
nostra13/Android-Universal-Image-Loader	2011	16.4k	Java	6.3k	35
nvbn/thefuck	2015	43.7k	Python	2.1k	123
octobercms/october	2013	8.5k	PHP	1.9k	303
omniauth/omniauth	2010	6.8k	Ruby	870	143
openai/gym	2016	16.6k	Python	4.4k	176
orhanobut/logger	2015	11.1k	Java	1.7k	10
overtrue/wechat	2015	7.8k	PHP	1.9k	98
pallets/flask	2010	44k	Python	12.2k	507
pandas-dev/pandas	2009	19.4k	Python	7.7k	1479
parcel-bundler/parcel	2017	31.3k	Javascript	1.4k	204
phalcon/cphalcon	2012	9.6k	PHP	1.7k	226
phanan/koel	2015	10.2k	PHP	1.2k	45
PhilJay/MPAndroidChart	2014	27k	Java	7k	67
php-ai/php-ml	2016	6.8k	PHP	947	28
PHPMailer/PHPMailer	2008	13.1k	PHP	7.2k	168
plataformatec/devise	2009	19.9k	Ruby	4.6k	541
plataformatec/simple_form	2009	7.3k	Ruby	1.2k	219
prettier/prettier	2916	31.4k	Javascript	1.7k	413
pypa/pipenv	2017	16.8k	Python	1.2k	276
rails/rails	2004	43.1k	Ruby	17.3k	3818
ramsey/uuid	2012	8.7k	PHP	315	59
rapid7/metasploit-framework	2005	16.3k	Ruby	8.1k	628
react-native-community/lottie-react-native	2016	11.2k	Java	1k	53
ReactiveX/RxAndroid	2013	17.9k	Java	2.8k	59

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
ReactiveX/RxJava	2012	38.6k	Java	6.5k	240
reactphp/react	2012	6.8k	PHP	672	29
ReactTraining/react-router	2014	35.9k	Javascript	7.3k	548
realm/realm-java	2012	10.4k	Java	1.6k	80
reduxjs/redux	2015	48.1k	Javascript	12.3k	673
resque/resque	2009	8.5k	Ruby	1.5k	207
resume/resume.github.com	2011	40.3k	Javascript	1k	48
roots/sage	2011	10k	PHP	2.8k	193
rubocop-hq/rubocop	2012	9.9k	Ruby	2k	596
ruby-grape/grape	2010	8.8k	Ruby	1k	319
ryanb/cancan	2009	6.3k	Ruby	839	54
scikit-learn/scikit-learn	2010	35k	Python	16.9k	1304
scrapy/scrapy	2008	32.8k	Python	7.6k	313
sebastianbergmann/phpunit	2006	13.8k	PHP	1.7k	358
Seldaek/monolog	2011	14.6k	PHP	1.5k	324
SeleniumHQ/selenium	2004	14.3k	Java	4.8k	429
Semantic-Org/Semantic-UI	2013	45.2k	Javascript	4.8k	190
serbanghita/Mobile-Detect	2012	8.6k	PHP	2.3k	83
serverless/serverless	2015	29.9k	Javascript	3k	571
sferik/rails_admin	2010	7k	Ruby	2k	357
Shopify/liquid	2008	7.1k	Ruby	931	121
signalapp/Signal-Android	2011	11.4k	Java	2.9k	183
sinatra/sinatra	2007	10.6k	Ruby	1.9k	361
skylot/jadx	2013	18.5k	Java	2k	31
slimphp/Slim	2010	9.8k	PHP	1.8k	202
socketio/socket.io	2004	46k	Javascript	8.5k	154
spree/spree	2008	9.7k	Ruby	4.2k	252
spring-projects/spring-boot	2012	36.8k	Java	24.1k	571
spring-projects/spring-framework	2008	29k	Java	19k	364
sqlmapproject/sqlmap	2008	14.1k	Python	3k	79
square/okhttp	2012	31.8k	Java	7k	182
square/picasso	2013	16.7k	Java	3.9	91
square/retrofit	2010	32k	Java	5.9k	125
StevenBlack/hosts	2012	12k	Python	1.1k	61
storybooks/storybook	2015	36.7k	Javascript	2.9k	677
stympy/faker	2007	7.7k	Ruby	1.9k	585
swiftmailer/swiftmailer	2007	7.8k	PHP	738	133
symfony/symfony	2010	20.6k	PHP	6.8k	1866
teamcapybara/capybara	2009	8.7k	Ruby	1.2k	259
Tencent/tinker	2016	13.6k	Java	2.7k	22
tensorflow/magenta	2016	13.1k	Python	2.5k	97
tensorflow/models	2016	52.6k	Python	31.7k	497

Table 4 (continued)

Repository	Since	Stars	Language	Forks	Contributors
the-control-group/voyager	2016	8k	PHP	1.9k	288
TheAlgorithms/Java	2016	13.5k	Java	5k	137
TheAlgorithms/Python	2016	38.4k	Python	10.9k	218
thedaviddias/Front-End-Checklist	2017	34.2k	Javascript	3.2k	82
thephpleague/flysystem	2013	9.3k	PHP	512	171
thepracticaldev/dev.to	2018	9.1k	Ruby	1k	187
thoughtbot/bourbon	2011	8.8k	Ruby	918	101
thoughtbot/factory_bot	2008	6.4k	Ruby	1.7k	187
thoughtbot/guides	2012	8k	Ruby	1.2k	98
thoughtbot/paperclip	2008	9k	Ruby	2k	371
tmuxinator/tmuxinator	2010	8.9k	Ruby	549	109
toddmotto/public-apis	2016	56.8k	Python	5.7k	475
tootsuite/mastodon	2016	17.6k	Ruby	3.1k	558
tornadoweb/tornado	2009	17.7k	Python	4.9k	304
trailofbits/algo	2016	13.1k	Python	1.1k	119
trekhleb/Javascript-algorithms	2018	47.9k	Javascript	6.7k	89
TryGhost/Ghost	2013	29.6k	Javascript	6.3k	314
twbs/bootstrap	2011	133k	Javascript	64.9k	1074
twbs/bootstrap-sass	2011	12.7k	Ruby	3.5k	95
tymondesigns/jwt-auth	2014	7.7k	PHP	968	65
typicode/json-server	2013	39.6k	Javascript	3.5k	61
udacity/fullstack-nanodegree-vm	2015	263	Python	11.3k	7
Valloric/YouCompleteMe	2012	19k	Python	2.1k	136
varvet/pundit	2012	6.4k	Ruby	489	92
vinta/awesome-python	2014	67.2k	Python	12.7k	306
vlucas/phpdotenv	2013	9.2k	PHP	439	47
vuejs/vue	2016	136k	Javascript	19.3k	274
walkor/Workerman	2013	7.4k	PHP	1.9k	49
webpack/webpack	2012	48.3k	Javascript	6k	516
wix/react-native-navigation	2016	10.2k	Java	2.2k	280
yarnpkg/yarn	2016	35.5k	Javascript	2.1k	496
yiisoft/yii2	2011	12.8k	PHP	6.7k	961
ytdl-org/youtube-dl	2008	50.3k	Python	8.5k	671
zeit/next.js	2016	36.7k	Javascript	4.2k	700
zxing/zxing	2013	22.4k	Java	8.1k	94

Appendix B

Table 5 presents the results of mining 250 issue repositories in detail.

Table 5 This table shows the frequencies of various elements in bug reports for different projects

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
30-seconds/30-seconds-of-code	9/184	23	49	0	17	32	11	1
achael/eht-imaging	1	0	2	2	0	0	0	0
activeadmin/activeadmin	34/541	203	527	317	130	101	17	4
adam-p/markdown-here	227/284	86	21	0	6	15	14	1
adobe/brackets	1169/3474	160	1148	4	923	238	253	2
ageitgey/face_recognition	302/431	0	175	114	41	26	44	1
airbnb/lottie-android	33/853	3	158	75	67	21	70	0
androidannotations/androidannotations	43/1564	0	253	150	67	52	3	3
angular/angular.js	129/2849	1622	1214	14	944	727	37	3
ansible/ansible	280/1349	13982	15790	3367	14834	612	97	83
apache/incubator-dubbo	331/2032	99	394	90	294	27	140	3
apache/incubator-echarts	3209/6916	566	1442	18	1397	28	1309	5
apache/incubator-zipkin	13/41	88	95	45	23	29	49	0
atech/postal	93/161	20	52	27	2	24	49	2
atom/atom	557/14128	2937	5693	816	5343	672	600	5
axios/axios	515/1168	86	138	50	37	56	99	9
aymericdamien/TensorFlow-Examples	129/53	0	21	20	0	2	3	0
babel/babel	625/5711	795	852	142	477	492	146	18
barryvdh/laravel-debugbar	107/120	0	28	7	6	15	27	0
barryvdh/laravel-ide-helper	294/257	0	28	6	4	18	24	3
bazelbuild/bazel	875/2251	1931	1724	217	1426	177	27	18
bcit-ci/CodeIgniter	45/2944	161	132	4	33	97	17	2
BetterErrors/better_errors	42/209	16	46	37	1	8	3	0
binux/pyspider	219/529	6	166	110	50	17	11	1
bobthecow/psysh	55/282	39	16	4	6	7	11	1
briannesbitt/Carbon	7/698	15	45	5	29	12	5	3
bumptech/glide	137/3205	395	538	414	71	87	129	5
CachetHQ/Cachet	127/1859	432	266	129	100	50	82	1
cakephp/cakephp	65/1704	30	379	68	88	231	31	10
capistrano/capistrano	18/509	77	220	141	56	42	0	2
carrierwaveuploader/carrierwave	169/1489	30	236	165	26	51	4	1
celery/celery	383/3441	376	1476	884	751	117	32	11
certbot/certbot	352/1629	470	837	606	69	184	11	4
chartjs/Chart.js	424/4137	1003	684	6	497	382	353	11
chrisbanes/PhotoView	116/451	39	59	44	7	10	10	0
CocoaPods/CocoaPods	178/6943	0	1778	1168	482	191	63	15
composer/composer	87/1280	663	577	35	163	395	45	14
daimajia/AndroidSwipeLayout	30/13	9	27	15	5	7	5	1
daimajia/AndroidViewAnimations	14/23	1	9	6	1	2	0	0
deeplearning4j/deeplearning4j	391/1780	967	533	306	138	108	122	6
deployphp/deployer	20/93	171	310	18	274	44	8	5
diaspora/diaspora	418/4141	1761	564	264	176	149	33	2
dingo/api	67/359	52	133	52	57	28	6	2

Table 5 (continued)

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
docker/compose	505/3891	511	970	461	488	109	24	17
doctrine/inflector	4/23	0	0	0	0	0	0	0
doctrine/instantiator	1/9	3	1	0	1	0	0	0
doctrine/lexer	Inf	0	0	0	0	0	0	0
Dogfalo/materialize	481/4486	279	533	22	390	222	176	7
donnemartin/system-design-primer	29/14	9	2	1	0	1	2	0
egulias/EmailValidator	21/83	24	2	0	1	1	1	0
elastic/elasticsearch	1736/18139	3510	5540	1719	3898	832	165	47
elastic/logstash	1366/3745	965	1071	506	457	200	41	7
encode/django-rest-framework	143/3023	311	812	158	608	120	35	3
EnterpriseQualityCoding/FizzBuzzEnterpriseEdition	134/27	1	15	2	0	13	0	0
erusev/parsedown	2/29	89	23	0	7	16	6	4
eugenp/tutorials	2/65	0	27	15	1	12	4	2
explosion/spaCy	70/1363	411	896	444	452	125	65	14
expressjs/express	111/2956	90	178	48	58	73	29	4
facebook/create-react-app	317/4174	400	1453	177	1243	181	338	12
facebook/fresco	79/1897	282	351	267	75	26	61	3
facebook/react	503/6964	369	1435	55	1212	204	217	10
facebook/react-native	219/7931	1319	4847	285	4329	436	1490	46
facebook/stetho	1/9	39	62	54	8	3	11	1
facebookresearch/Detectron	213/571	6	256	139	130	24	36	0
faif/python-patterns	9/47	1	3	1	0	2	1	0
fastlane/fastlane	171/9137	522	2085	1754	69	332	140	24
filp/whoops	42/251	0	18	3	0	15	4	0
fluent/fluentd	197/960	106	290	232	52	36	16	3
FortAwesome/Font-Awesome	4769/9638	385	456	6	143	312	1079	4
freeCodeCamp/devdocs	77/647	102	51	14	21	17	24	0
freeCodeCamp/freeCodeCamp	75/4444	261	7730	26	7465	366	944	9
FriendsOfPHP/Goutte	105/131	0	9	4	3	2	3	0
FriendsOfPHP/PHP-CS-Fixer	234/1373	247	77	9	23	48	8	4
gatsbyjs/gatsby	419/6047	534	2005	148	1611	564	442	27
getgrav/grav	103/556	219	121	24	39	59	75	5
getredash/redash	43/148	220	499	120	407	36	102	2
getsentry/sentry	68/253	73	489	291	152	72	184	0
github/linguist	5/417	16	62	17	7	38	30	0
gollum/gollum	14/449	212	114	66	24	25	4	0
google/ExoPlayer	294/4849	841	1334	756	524	171	112	3
google/gson	8/19	2	345	92	273	30	6	1
google/guava	371/1104	0	249	27	90	136	7	0
google/python-fire	23/31	13	14	8	1	6	1	1
google-research/bert	11/8	0	71	42	13	21	23	0
GoogleChrome/puppeteer	376/2539	186	1271	227	1115	83	226	26
greenrobot/greenDAO	31/181	35	67	54	6	12	15	1

Table 5 (continued)

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
gulpjs/gulp	17/1707	27	116	47	20	52	22	3
guzzle/guzzle	127/597	24	194	30	138	40	17	3
h5bp/html5-boilerplate	1/1142	38	78	1	12	65	0	0
hakimel/reveal.js	370/1269	49	80	3	35	42	19	1
hashicorp/vagrant	199/3767	1600	2657	869	1738	256	54	14
hdodenhof/CircleImageView	9/275	3	35	29	2	5	10	0
HelloZeroNet/ZeroNet	535/1064	98	231	82	118	41	56	0
home-assistant/home-assistant	1007/8670	143	4607	2641	2840	158	545	19
Homebrew/brew	3/707	86	647	239	406	111	19	13
Homebrew/homebrew-cask	1/222	18	1090	956	31	138	20	3
huge-success/sanic	21/223	42	119	91	14	21	11	1
huginn/huginn	288/1207	20	223	160	9	65	20	2
iluwatar/java-design-patterns	181/263	40	9	2	0	7	0	0
imathis/octopress	179/808	70	140	101	16	30	0	1
impress/impress.js	46/379	5	28	0	16	12	0	0
Intervention/image	229/500	8	29	6	4	19	19	1
ipython/ipython	144/611	930	1118	679	266	240	73	8
JakeWharton/butterknife	39/484	9	103	65	12	30	32	4
jakubroztocil/httpie	13/41	81	79	61	8	13	9	0
javan/whenever	55/463	25	87	69	2	16	0	0
jekyll/jekyll	67/3872	23	833	318	421	136	25	2
jfeinstein10/SlidingMenu	263/381	6	72	43	13	21	1	0
jordansissel/fpm	14/13	64	188	146	24	34	2	0
josephmisiti/awesome-machine-learning	1/57	0	3	0	1	2	0	0
jquery/jquery	13/358	103	145	5	84	59	38	1
juliangarnier/anime	68/377	23	34	1	25	8	15	2
kaminari/kaminari	5/146	57	66	42	16	11	1	1
kennethreitz/requests	25/391	137	655	512	107	94	27	6
keon/algorithms	14/29	7	5	1	0	4	3	0
keras-team/keras	2323/6809	112	2479	1044	1324	389	212	21
kilimchoi/engineering-blogs	7/59	0	0	0	0	0	0	0
Konloch/bytecode-viewer	29/154	35	37	27	3	8	3	0
laravel/framework	29/4011	346	3704	197	3255	431	176	49
lgvalle/Material-Animations	1/2	0	2	1	0	1	0	0
LMAX-Exchange/disruptor	6/187	2	33	10	16	8	3	2
localstack/localstack	410/489	199	256	210	38	26	11	4
lodash/lodash	0	227	154	12	49	94	40	2
loopj/android-async-http	243/799	0	134	107	2	28	2	0
Maatwebsite/Laravel-Excel	27/1832	18	418	18	378	42	104	8
magento/magento2	557/6754	3604	9383	664	8826	445	1787	15
mame/quine-relay	1/8	0	3	0	0	3	0	0
matomo-org/matomo	1717/8490	4102	920	85	439	416	301	3
matterport/Mask_RCNN	857/483	0	203	143	4	61	99	5

Table 5 (continued)

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
meteor/meteor	290/7729	553	994	250	506	300	90	16
Microsoft/vscode	2743/31334	17233	28328	503	25881	3228	10134	119
middleman/middleman	123/1426	56	366	226	130	41	1	2
mikepenz/MaterialDrawer	8/2269	120	180	100	40	45	89	4
minimaxir/big-list-of-naughty-strings	43/24	0	7	0	1	6	1	0
mitmproxy/mitmproxy	53/425	368	1002	456	777	42	51	2
mockery/mockery	52/437	62	36	8	15	15	2	1
moment/moment	47/628	252	550	15	478	73	47	8
monicahq/monica	332/717	199	65	15	22	29	103	0
mperham/sidekiq	12/3073	0	575	414	69	128	31	13
mrdoob/three.js	629/8187	900	571	28	172	378	322	8
mui-org/material-ui	281/8286	1286	3057	74	2716	868	779	21
mybatis/mybatis-3	95/763	113	259	73	190	21	16	1
NARKOZ/hacker-scripts	9/8	0	2	0	1	1	0	0
NationalSecurityAgency/ghidra	241/276	258	176	6	163	14	109	1
netty/netty	388/4203	231	1332	565	776	112	46	1
nextcloud/server	1961/5781	3389	4114	546	3891	206	828	26
nicolargo/glances	107/955	345	284	258	22	23	30	3
nikic/PHP-Parser	41/343	0	33	8	7	18	2	1
nodejs/node	337/4612	645	1727	799	599	419	192	40
nostra13/Android-Universal-Image-Loader	273/472	89	181	145	10	31	2	0
nvbn/thefuck	37/80	3	156	76	60	48	11	0
octobercms/october	163/1164	717	779	23	696	85	179	13
omniauth/omniauth	29/263	15	87	57	19	17	2	0
openai/gym	175/689	1	191	136	21	43	22	1
orhanobut/logger	44/129	13	11	6	0	5	7	1
overtrue/wechat	22/719	8	17	17	0	0	72	3
pallets/flask	1/52	28	275	204	37	53	13	4
pandas-dev/pandas	963/4034	4242	1798	1144	354	377	138	16
parcel-bundler/parcel	321/715	779	598	126	387	349	151	9
phalcon/cphalcon	48/2011	630	351	82	184	100	19	5
phanan/koel	8/79	26	94	9	67	23	11	1
PhilJay/MPAndroidChart	201/389	82	314	174	40	112	342	2
php-ai/php-ml	26/35	0	10	4	0	6	7	0
PHPMailer/PHPMailer	29/1306	8	201	13	140	50	22	1
plataformatec/devise	5/764	147	586	416	87	114	11	5
plataformatec/simple_form	17/1099	61	76	39	15	25	8	0
prettier/prettier	583/2822	1142	220	41	126	60	96	41
pypa/pipenv	281/2355	332	877	733	118	80	46	9
rails/rails	371/12260	0	4799	1535	3541	361	86	30
ramsey/uuid	27/89	15	11	0	2	9	2	0
rapid7/metasploit-framework	637/2466	1032	1265	531	892	65	91	1
react-native-community/lottie-react-native	29/151	23	63	10	46	8	29	6

Table 5 (continued)

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
ReactiveX/RxAndroid	0	0	35	22	3	12	3	0
ReactiveX/RxJava	14/1333	237	347	164	95	102	13	8
reactphp/react	0	14	10	1	1	8	0	0
ReactTraining/react-router	7/988	141	871	25	731	138	90	10
realm/realm-java	425/3378	589	1161	568	694	102	38	14
reduxjs/redux	26/1615	16	209	8	113	96	33	1
resque/resque	34/769	113	104	79	12	19	2	0
resume/resume.github.com	29/54	11	6	0	4	2	0	0
roots/sage	17/1124	40	123	7	93	29	4	5
rubocop-hq/rubocop	225/2999	615	1312	341	1000	170	13	10
ruby-grape/grape	59/244	313	121	87	15	25	2	0
ryanb/cancan	68/215	15	84	50	3	33	0	0
scikit-learn/scikit-learn	1253/4945	930	1748	628	1111	248	61	12
scrapy/scrapy	59/149	141	384	282	42	79	17	4
sebastianbergmann/phpunit	69/2329	170	187	55	71	68	25	5
Seldaek/monolog	7/85	35	37	12	5	21	2	1
SeleniumHQ/selenium	365/5286	0	2941	251	2701	162	157	16
Semantic-Org/Semantic-UI	781/5176	1190	487	22	322	149	180	5
serbanghita/Mobile-Detect	36/109	78	39	0	0	39	5	1
serverless/serverless	497/3000	793	363	209	72	93	45	11
sferik/rails_admin	340/1787	151	258	192	24	55	14	0
Shopify/liquid	27/106	46	36	16	10	10	4	2
signalapp/Signal-Android	282/6443	104	2417	224	2064	597	345	2
sinatra/sinatra	65/638	86	98	64	18	21	0	0
skylot/jadx	39/188	31	29	21	4	5	33	0
slimphp/Slim	2/713	54	95	28	15	54	6	1
socketio/socket.io	377/2333	108	343	20	265	219	26	2
spree/spree	33/1186	0	838	394	404	152	31	2
spring-projects/spring-boot	381/13093	1375	2164	802	1187	298	110	30
spring-projects/spring-framework	731/17331	4006	1422	323	505	656	13	2
sqlmapproject/sqlmap	4/281	828	1611	1451	226	80	26	1
square/okhttp	85/1228	508	611	492	107	60	33	5
square/picasso	27/214	8	207	161	25	30	9	0
square/retrofit	29/1051	19	295	230	31	45	19	5
StevenBlack/hosts	28/307	17	42	23	3	16	19	1
storybooks/storybook	111/1010	706	1106	105	992	87	405	15
stympy/faker	47/334	35	49	30	13	9	9	0
swiftmailer/swiftmailer	151/223	0	87	29	37	23	3	3
symfony/symfony	635/11437	2787	1747	109	1225	560	268	35
teamcapybara/capybara	1/1302	78	304	156	146	44	7	0
Tencent/tinker	69/440	63	198	197	0	3	34	0
tensorflow/magenta	173/381	0	132	109	5	25	28	0
tensorflow/models	1269/2959	112	1904	913	1226	141	207	10

Table 5 (continued)

Repository	#O/C	#bugA	#bugB	#ST	#RS	#FS	#UC	#C
the-control-group/voyager	66/773	470	1289	6	1239	68	350	7
TheAlgorithms/Java	1/10	2	6	0	0	6	1	1
TheAlgorithms/Python	35/86	3	5	2	0	3	3	0
thedaviddias/Front-End-Checklist	3/88	14	7	0	1	6	7	0
thephpleague/flysystem	9/520	5	52	15	18	21	2	4
thepracticaldev/dev.to	11/35	332	317	9	274	39	291	3
thoughtbot/bourbon	1/275	18	45	8	24	15	0	0
thoughtbot/factory_bot	18/767	25	129	88	15	30	3	1
thoughtbot/guides	1/15	0	3	0	0	3	1	0
thoughtbot/paperclip	15/1801	138	243	170	24	58	3	1
tmuxinator/tmuxinator	80/321	41	50	28	13	10	1	0
toddmotto/public-apis	8/95	0	10	0	1	9	2	0
tootsuite/mastodon	1175/3353	779	741	169	270	325	412	8
tornadoweb/tornado	142/1301	0	313	251	39	51	9	2
trailofbits/algo	4/109	77	478	46	454	28	22	1
trekhleb/Javascript-algorithms	33/68	13	9	0	4	5	5	0
TryGhost/Ghost	81/5305	1424	1659	90	1459	215	205	5
twbs/bootstrap	329/18145	200	1185	18	473	708	601	13
twbs/bootstrap-sass	3/811	39	71	36	15	23	4	1
tymondesigns/jwt-auth	397/892	6	158	21	101	42	24	2
typicode/json-server	199/183	3	42	21	7	15	15	5
udacity/fullstack-nanodegree-vm	3/2	0	4	1	3	3	3	0
Valloric/YouCompleteMe	42/2609	5	927	376	546	181	62	1
varvet/pundit	8/329	4	48	29	1	18	1	1
vinta/awesome-python	71/69	0	5	0	1	4	0	1
vlucas/phpdotenv	0	0	11	1	3	7	0	0
vuejs/vue	67/2606	330	3150	47	2991	176	146	26
walkor/Workerman	8/255	10	10	6	0	4	10	1
webpack/webpack	442/5893	811	2404	176	2222	185	195	21
wix/react-native-navigation	69/1738	119	1549	77	1461	75	459	18
yarnpkg/yarn	347/674	571	2627	309	2447	80	108	31
yiisoft/yii2	14/249	1638	2214	189	1763	376	87	28
ytdl-org/youtube-dl	2352/14395	474	9337	6980	127	4517	97	10
zeit/next.js	205/3877	208	1336	254	1064	324	319	16
zxing/zxing	2/223	40	123	48	53	29	70	1

O/C indicates the ratio between open issues and closed one. **bugA** indicates that the issues is originally labeled as a bug, whereas **bugB** indicates that IMaChecker detected the issue as a bug. **ST** indicates Stack Traces, **RS** indicates Reproducing Steps, **FS** indicates Fix Suggestions, **UC** shows User Content, and **C** shows Code

References

Adolph S, Hall W, Kruchten P (2011) Using grounded theory to study the experience of software development. Empir Softw Eng 16(4):487–513

Anvik J, Hiew L, Murphy GC (2005) Coping with an open bug repository. In: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, ACM, pp 35–39

Barriball KL, While A (1994) Collecting data using a semi-structured interview: a discussion paper. J Adv Nursing-Institut Subscript 19(2):328–335

Basili VR, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25(4):456–473

Borges H, Valente MT, Hora A, Coelho J (2015) On the popularity of github applications: a preliminary note. arXiv:1507.00604

Braun V, Clarke V (2006) Using thematic analysis in psychology. Qual Res Psycho 3(2):77–101

Chen N, Kim S (2015) Star: stack trace based automatic crash reproduction via symbolic execution. IEEE Tr Sw Eng 41(2):198–220

Creswell JW, Creswell JD (2017) Research design: qualitative, quantitative, and mixed methods approaches. Sage Publications, California

De Vaus D, de Vaus D (2013) Surveys in social research. Routledge, Abingdon

Fink A (2003) The survey handbook. Sage, Newbury Park

Gibbs GR (2007) Thematic coding and categorizing. analyzing qualitative data. Sage, London, pp 38–56

developer.github (2015) https://developer.github.com/v3/issues/, accessed on 2015-05-01

Github (2019a) https://github.com/netty/netty/issues/8285, accessed on 2019-06-20

Github (2019b) https://github.com/axios/axios/issues/376, accessed on 2019-06-20

Github (2019c) https://github.com/axios/axios/issues/246, accessed on 2019-06-20

Github (2019d) https://github.com/apache/dubbo/issues/1500, accessed on 2019-06-20

Github (2019e) https://github.com/apache/dubbo/issues/3236, accessed on 2019-06-20

Github (2019f) https://github.com/activeadmin/activeadmin/issues/2623, accessed on 2019-06-20

Github (2019g) https://github.com/activeadmin/activeadmin/issues/3185, accessed on 2019-06-20

Github (2019h) https://github.com/activeadmin/activeadmin/issues/4650, accessed on 2019-06-20

Github (2019i) https://github.com/angular/angular.js/issues/16697, accessed on 2019-06-20

Github (2019j) https://github.com/ageitgey/face_recognition/issues/854, accessed on 2019-06-20

Github (2019k) https://github.com/airbnb/lottie-android/issues/1202, accessed on 2019-06-20

Github (2019l) https://github.com/barryvdh/laravel-debugbar/issues/237, accessed on 2019-06-20

Github (2019m) https://github.com/activeadmin/activeadmin/issues/4250, accessed on 2019-06-20

Glaser BG, Holton J (2004) Remodeling grounded theory. In: Forum qualitative sozialforschung/forum: qualitative social research, vol 5

Hooimeijer P, Weimer W (2007) Modeling bug report quality. In: Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering, ACM, pp 34–43

Hove SE, Anda B (2005) Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11Th IEEE international software metrics symposium (METRICS'05), IEEE, pp 10–pp

Jacob SA, Furgerson SP (2012) Writing interview protocols and conducting interviews: Tips for students new to the field of qualitative research. Qual Rep 17(42):1–10

Kitchenham BA, Pfleeger SL (2002) Principles of survey research: part 3: constructing a survey instrument. ACM SIGSOFT Softw Eng Notes 27(2):20–24

Likert R (1932) A technique for the measurement of attitudes. Archives of psychology

Moghaddam A (2006) Coding issues in grounded theory. Issues Educ Res 16(1):52–66

Myers MD, Newman M (2007) The qualitative interview in is research: examining the craft. Inform Organ 17(1):2–26

Nayrolles M, Hamou-Lhadj A, Tahar S, Larsson A (2015) Jcharming: a bug reproduction approach using crash traces and directed model checking. In: 2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER), IEEE, pp 101–110

Pfleeger SL, Kitchenham BA (2001) Principles of survey research: part 1: turning lemons into lemonade. ACM SIGSOFT Softw Eng Notes 26(6):16–18

Schroter A, Schröter A., Bettenburg N, Premraj R (2010) Do stack traces help developers fix bugs? In: 2010 7th IEEE working conference on mining software repositories (MSR 2010), IEEE, pp 118–121

Soltani M, Panichella A, Van Deursen A (2018) Search-based crash reproduction and its impact on debugging. IEEE Trans Softw Eng

The state of the octoverse (2017) Https://octoverse.github.com/2017/, accessed on 2019-05-012019

The state of the octoverse (2018) Top programming languages of. Https://github.blog/2018-11-15-state-of-the-octoverse-top-programming-languages/, accessed on 2019-05-01
The state of the octoverse (2019) Https://octoverse.github.com/, accessed on 2019-05-01
Vargha A, Delaney HD (2000) A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J Educ Behav Stat 25(2):101–132
Weiss C, Premraj R, Zimmermann T, Zeller A (2007) How long will it take to fix this bug? In: Fourth international workshop on mining software repositories (MSR'07: ICSE workshops 2007), IEEE, pp 1–1
Zeng H, Rine D (2004) Estimation of software defects fix effort using neural networks. In: Proceedings of the 28th annual international computer software and applications conference, 2004. COMPSAC 2004. IEEE, vol 2, pp 20–21
Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C (2010) What makes a good bug report? IEEE Trans Softw Eng 36(5):618–643

Mozhan Soltani is a PhD student at Leiden University. Her research interests include software processes, software testing, and requirements engineering.

Felienne Hermans Felienne is associate professor at the Leiden Institute of Advanced Computer Science at Leiden University, where she heads the PERL research group, focused on programming education. Felienne was also one of the founders of the Joy of Coding conference, which she organized for 6 years. Since 2016, she has been a host at SE radio, one of the most popular software engineering podcasts on the web.

Thomas Bäck is head of the Natural Computing Research Group and Director of Education at the Leiden Institute of Advanced Computer Science (LIACS). He received his PhD in Computer Science from Dortmund University, Germany, in 1994. He has been Associate Professor of Computer Science at Leiden University since 1996 and full Professor for Natural Computing since 2002.