Understanding Key Features of High-impact Bug Reports

Md. Rejaul Karim*, Akinori Ihara*, Xin Yang†, Hajimu Iida*, and Kenichi Matsumoto*
*Nara Institute of Science and Technology (NAIST), Japan
†Osaka University, Japan
Email: {rejaul.karim.qw4,akinori-i,matumoto}@is.naist.jp, xinyang@ist.osaka-u.ac.jp, iida@itc.naist.jp

***Abstract*—Nowadays, software projects are receiving bug reports on a daily basis. Developers cannot treat all the bugs in the same priority since some bugs can significantly affect software development process and the quality of products. Previous studies defined these bugs as High Impact Bug (HIB) and they found that HIB should be fixed quicker than other bugs in software development. However, fixing a HIB sometimes become complicated because the low-quality bug reports can be delay the bug fixing. In this study, we investigate what information is essential when reporting a HIB report. As a case study, we manually examine the HIB reports and perform both qualitative and quantitative analysis in Apache Camel project. Our main findings include: (1) we find four types of features are the most requested information from developers when they fix HIB; (2) the requested additional information significantly influences bug fixing time.**

I. Introduction

One of the key tasks in software development process is fixing bugs according to the bug reports submitted by developers, testers, or end-users. In a large and evolving software system, the large amount of bug reports typically exceed the available project resources. Accordingly, some bugs might be dealt with a long term delay or not at all [1].

To avoid missing bugs which impact to product and development process, we focus on High Impact Bug (HIB). Existing studies defined six types of HIB and they found that HIB should be fixed quicker than other bugs in software development [2][3][4][5][6][7]. For example, security bug (one of the type of HIB) should be fixed faster than other non-HIB because it allows unauthorized access to the system. However, fixing a HIB sometimes become complicated because of the low-quality bug reports. Davies et al. found that bug reports are neither complete nor accurate through a case study on four big scale and successful open source projects (Eclipse, Firefox, Apache HTTP, and Facebook API) [8]. Davies also found that developers spend lots of time to collect additional features to fix bugs.

In particular, most bug reporters work voluntarily in OSS projects. To make their work easy, bug reporting should be simple as much as possible. Joel Spolsky noted *"I have always felt that if you can make it 10 percent easier to fill in a bug report, you will get twice as many bug reports"* [9]. It is difficult for novice developers or end-users to know which features should be included in the bug reports. In addition, there is no standard guidelines for users to make a good bug report. In this study, we focus on HIB reports and conduct an empirical study to understand how we can make a good HIB report by providing minimum features. We manually analyzed six types of HIB reports of Apache Camel project and checked the contents of each bug report. We also analyzed the conversations between developers and reporters to understand the features that have been requested when developers fix bugs. We address the following three research questions:

RQ1: What are the features that reporters provide in each type of high-impact bugs?
The previous research revealed useful 10 features for developers to fix a common bug [10] based on a survey of experienced developers. Another case study found that different types of bugs (e.g. Performance and Security bugs) differ from each other [11]. Our hypothesis is that key features may also differ among HIB to fix.

RQ2: Which features did developers often request to provide in each type of high-impact bugs?
Reporters do not know which features are useful when fix HIB. Hence, developers often ask additional features of bug reports to the reporters. We address this research question to understand what features developers often request in each type of HIB reports.

RQ3: Does requested features influence bug fixing time?
High-quality bug reports would be crucial to fix HIB in time. Developers' requests for additional features may increase bug-fixing time. We explore that how significantly developers' requested additional features affect bug-fixing time.

Our findings show that Test Cases, Code Examples, Steps to Reproduce, and Stack Traces are the most additionally requested features and the developers' requests significantly increase bug fixing time.

II. Background

In this section, we first briefly introduce high impact bugs. Then, we describe our motivation of this research.

A. High-impact Bug (HIB)

A bug is considered as a high-impact bug (HIB) if it highly impact on software processes, product, or end-users. Software engineering researchers have introduced different HIB based on their impact [2][3][4][5][6][7]. We list the different types of HIB as follows:

978-1-5090-6699-5/17 $31.00 © 2017 IEEE
DOI 10.1109/IWESEP.2017.17

1) **Surprise bugs:** A surprise bug [3] is a new concept on software bugs . It can disturb the workflow and/or task scheduling of developers, since it appears in unexpected timing (e.g., bugs detected in post-release) and locations (e.g., bugs found in files that are rarely changed in pre-release).
2) **Dormant bugs:** A dormant bug [5] is also a new concept on software bugs and defined as a bug that was introduced in one version (e.g., Version 1.1) of a system, yet it is Not reported until after the next immediate version (i.e., a bug is reported against Version 1.2 or later). Another research fournd that 33% of the reported bugs in Apache Software Foundation (ASF) projects were dormant bugs [12].
3) **Blocking bugs:** A blocking bug is a bug that blocks other bugs from being fixed [13]. It often happens if a dependency relationship exists among software components. Since a blocking bug inhibit developers from fixing other dependent bugs, it has a high impact on developers task scheduling.
4) **Security bugs:** A security bug [14] can raise a serious problem which often impacts on uses of software products directly. It exploits to gain unauthorized access or privileges in the systems. In general, security bugs are supposed to be fixed as soon as possible.
5) **Performance bugs:** A performance bug [15] is defined as programming defects that cause significant performance degradation. The performance degradation contains poor user experience, lazy application responsiveness, lower system throughput, and needles waste of computational resources.
6) **Breakage bugs:** A breakage bug [3] is a functional bug which is introduced into a product because the source code is modified to add new features or to fix existing bugs. A breakage bug can cause usable functions in old versions unusable after releasing new versions.

B. Motivation of this research

Developers rely on the content of bug reports to locate and fix a bug. A well-written and informative bug report would be helpful to fix a bug for developers. For example, a bug report[1] was submitted in the Jira issue tracking system. The bug report related to a performance problem. The reporter describes in details what he saw happen, what he expected to happen, sample codes etc. The reporter also attached Test Cases in the report to test after the developer fixed. The developer fixed the bug on the same day of reporting.

However, unfortunately, all bug reports are not always well informative. For example, the bug report[2] was submitted in the same repository. The bug related to Breakage as well as Surprise bug. This bug report was created on December 10, 2012. The reporter provided what he saw happen, stack traces, and some other information. When the bug report assigned to the developer, he requested to the reporter to provide additional information like this: *"Can you submit a small test case for us the reproduce the error?"*. 2 months and 17 days (on February 27, 2013) later, the reporter responded and provided Steps to Reproduce. Finally, developer fixed the bug after 8 days. We can know that this example describes the quality of bug reports negatively influenced on bug fixing.

[1]https://issues.apache.org/jira/browse/CAMEL-3540
[2]https://issues.apache.org/jira/browse/CAMEL-5860

TABLE I
NO.OF BUG REPORTS IN TERMS OF HIGH-IMPACT BUGS

High-impact bug	No. of bug reports
Surprise	128
Dormant	69
Blocker	7
Security	14
Performance	51
Breakage	39
Total	308

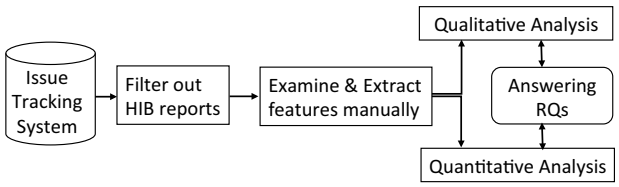

Fig. 1. Overall HIB reports analysis procedure

To help bug fixing, we need to understand what information the reporters should submit in the HIB reports. However, there is no standard guideline for the reporters to file HIB report correctly. Therefore, we motivated to do an empirical case study on HIB reports to investigate in details and to reveal insightful information about what makes a good HIB report. Consequently, we can make a valuable contribution formulating a standard guideline for reporters on how to write HIB report more accurately. By achieving our study, our outcome will be helpful to develop a tool to assess the quality and to suggest what information should be included to improve the quality of HIB report.

III. CASE STUDY DESIGN

A. Target Dataset

We conduct a case study on Apache Camel bug dataset that is shared by Ohira et al. [7]. The dataset was created by manually reviewing four thousand bug reports for four open source projects (Ambari, Camel, Derby and Wicket). As kick-off study, we focus on Camel project to understand key features to report HIB. Table I shows the statistics of analyzed bug reports. To address our three research questions, we first filtered out HIB reporters from Jira bug tracking system based on [7] for the Apache Camel project. Then, we performed qualitative and quantitative analysis to address our research questions. The figure 1 describes overall analysis procedure.

TABLE II
THE LIST OF TOP 10 MOST IMPORTANT FEATURES TO FIX THE BUGS

Name of Features	Short Description
Steps to Reproduce (STR)	A clear set of instructions that the developer can use to reproduce the bug on their own machine.
Stack traces (ST)	A stack trace produced by the application, most often when the bug is reporting a crash in the application.
Test Cases (TC)	One or more test cases that the developer can use to determine when they have fixed the bug
Observed behaviour (OB)	What the user saw happen in the application as a result of the bug
Screenshots (SS)	A screenshot of the application while the bug is occurring
Expected behaviour (EB)	What the user expected to happen, usually contrasted with Observed Behaviour
Code Examples (CE)	An example of some code which can cause the bug
Summary (S)	A short (usually one-sentence) summary of the bug
Version (V)	What version of the application the user was using at the time of the error
Error reports (ER)	An error report produced by the application as the bug occurred

1) Qualitative Analysis: In our case study, we manually analyzed HIB reports. Typically, a bug report contains a combination of structured (e.g., version, severity, environment, reporter) and unstructured information (e.g., summary, Steps to Reproduce, Observed Behavior). Software engineering researchers have defined these structured and unstructured information as features. Bettenburg et al. [10] surveyed among 156 experienced developers of Apache project, Eclipse project and Mozilla project to examine what features developers expect in a bug report. Based on the feedback from the developers, they revealed the top 10 most important features that developers found useful for bug fixing. Table II shows the top 10 most important features to fix a bug. Among these 10 features, summary and version are equal important for all kind of HIB reports. So, we excluded these two features and included one features named Environment (EN) in our study.

However, we believe that all features are not equally important for all kind of HIB report. That means features that developers find useful on bug fixing may vary among different type of HIB. In addition, bug reporters face complexities to provide some features in each HIB report. For instance, bug reporters are not able to provide Stack Traces in each HIB report because it does not always produces especially for the performance bug. Therefore, we need to understand the key features for each type of HIB so that reporters can make good quality bugs by providing minimum features. In order to reveal the key features for each type of HIB report, we have conducted a qualitative analysis. Our qualitative is comprised mainly of two phases.

In the first phase for RQ1, we manually examine each HIB report and extract the reported features from the bug reports. Actually, the structured features are easy to extract automatically. On the other hand, the unstructured 10 features are not so easy to extract because those features are provided with natural language text in the description field. Hence, we conduct a manual examination and observe each reported feature. In detail, we observed what features developer provided and how they provided and what was the missing features for each HIB report.

In the second phase for RQ2, we mainly examined developers activities on bug fixing. During bug fixing, developers and reporters communicate with each other for the various purposes such as clarification, gather missing features, status inquiry. Usually, when a bug report is assigned to a developer to fix the bug, developer start work. If the bug report are not well-written and does not contain sufficient information to localize and to fix the bug, developer pause his work and request to provide the missing features. Then, reporter provides the requested feature. This type of communication kills the developer's valuable time. The idea behind analyzing additional request is to understand developer need to fix each type of HIB. Mining frequently developers request habit for additional features can be an important factor to understand key features for each type of HIB.

2) Quantitative Analysis: During our qualitative analysis for RQ1 and RQ2, we found that sometimes reporters did not provide some features at initial bug report submission. In order to collect the missing features, developers make additional request to the reporters to provide them. We believe that it affects on overall bug fixing. Every bug reporter and project manager assume some level of delay for the bug fix. However, the excessive delay is not tolerable to fix a bug, especially for HIB.

For this reason, we conduct a quantitative analysis to understand whether the request for additional features significantly affects on bug fixing. In detail, we compare the bug fixing time difference between HIB reports with additional feature request (With Request) and HIB reports with no additional feature request (Without Request). The idea behind doing this is to make sense reporters about the aftermath of submitting informative or less informative HIB report. If it significantly affects on bug fixing then reporters will be more careful to file HIB report as well as researchers will be encourage to do research on how to mitigate the effect.

IV. RESULTS

RQ1: What are the features that reporters provide in each type of high-impact bugs?

To answer RQ1, we carefully examined features in bug reports by reading manually. Figure 2 shows how often HIB reports contains each key features.

We found average percentages of reported features in Performance, security, Breakage, Surprise, Dormant, and Blocker bug reports are 33%, 33%, 32%, 29%, 27%, and 31% respectively (red line). In particular, *Observation behaviour* is the most frequent features in HIB reports. Also, *Expected behaviour* and *Code example* are the next frequent features

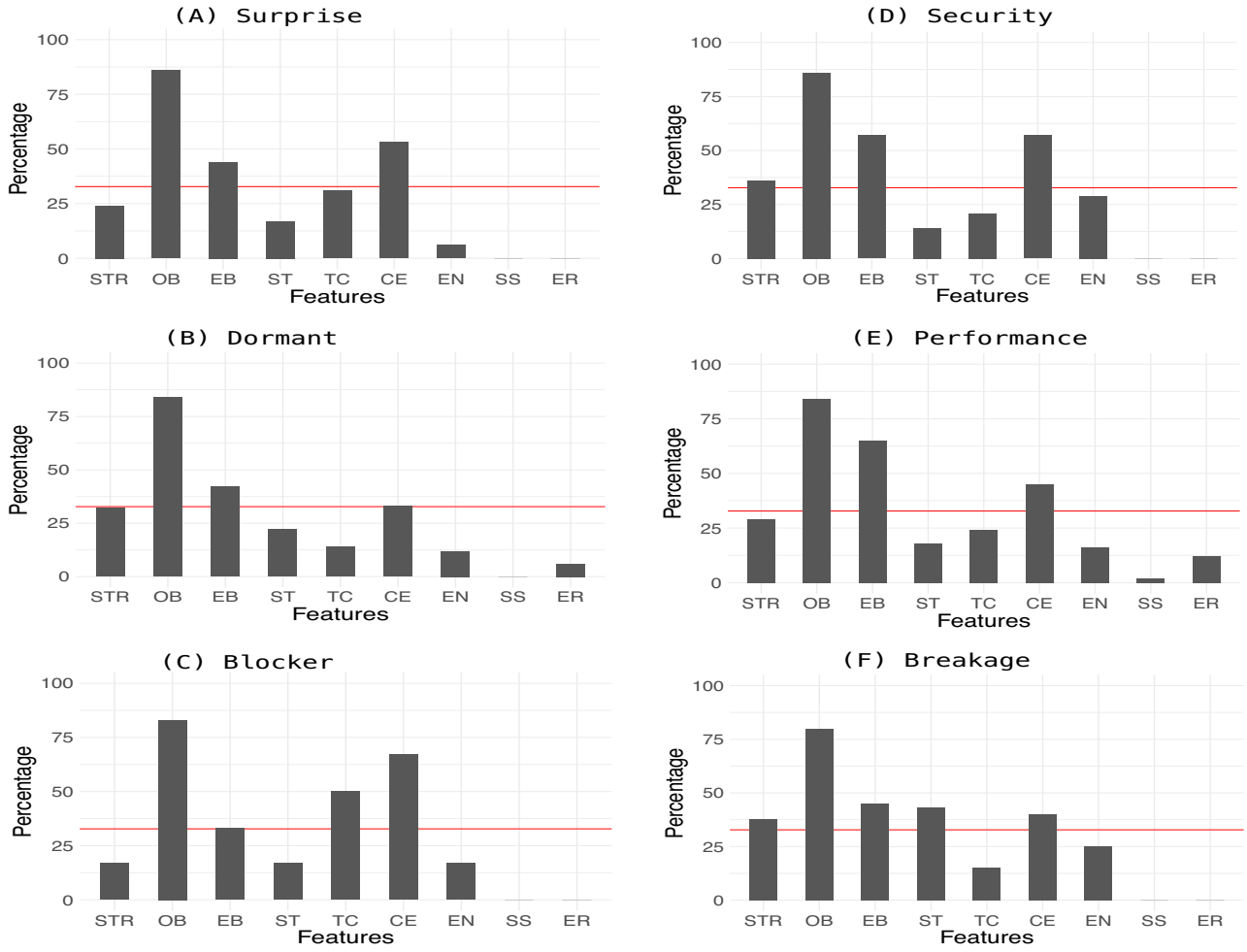

Fig. 2. Observed reported features in bug reports in terms of HIB

in HIB reports. On the other hand, *Screenshot* and *Error report* are the least frequent features in HIB reports. The other features are different ranking depends on the type of HIB. For example, while *Test code* is often submitted for only Blocker bug, it is often not submitted for the other type of bug.

RQ2: Which features did developers often request to provide in each type of high-impact bugs?

To answer RQ2, we examined additional features which developers often asked a reporter to submit. We found that developers made additional request for around 19% HIB reports. Table III shows the percentage of additionally requested features in each type of HIB. The percentage of additional request for the feature, $x_i = \frac{\#Request\ for\ x_i}{\sum_{i=1}^{i=n}(\#Request\ for\ x_i)} \times 100\%$ where n is the total number of features. We found that developers often asked to *Steps to reproduce*, *Test case* and *Code example* for any types of HIB. For *Steps to reproduce*, 50% security bug report was asked. For *Test case*, approximately 50% of Performance, Breakage, Surprise and Dormant bugs reports were asked. For *Code example*, 50% security bug reports was asked. In our examination, we found that developers did not make any request to provide additional feature for the Blocker bug reports. Blocker bug might be required less features to fix than other HIB.

TABLE III
ADDITIONALLY REQUESTED FEATURES DURING BUG FIXING IN THE HIGH-IMPACT BUG REPORTS

Features	High-impact bugs					
	Performance	Security	Breakage	Surprise	Dormant	Blocker
STR	15%	50%	9%	14%	0%	0%
OB	0%	0%	0%	0%	0%	0%
EB	0%	0%	0%	0%	0%	0%
ST	8%	0%	0%	14%	13%	0%
TC	54%	0%	55%	52%	50%	0%
CE	23%	50%	27%	19%	38%	0%
EN	0%	0%	0%	0%	0%	0%
SH	0%	0%	0%	0%	0%	0%
ER	0%	0%	0%	0%	0%	0%

RQ3: Does requested features influence bug fixing time?

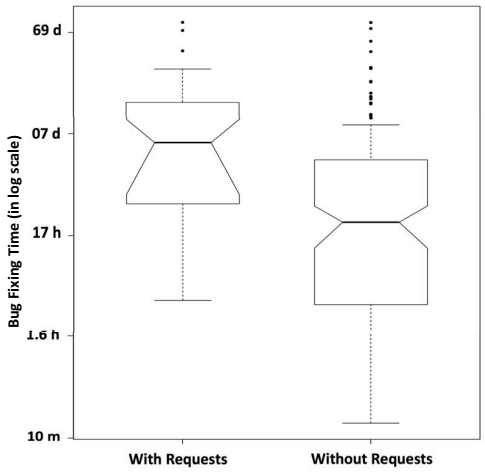

Fig. 3. Distribution of bug fixing time between additional features request and no-request groups

To answer RQ3, we analyze the impact of bug fix due to additional request for HIB reports. First, we classified HIB reports into two groups; for additional features request (With Request) and no additional features request (Without Request). Then, we measured bug-fixing time which is the bug report created date to fixed date. Figure 3 shows the distribution of bug fixing time among additional requested group and no requested group. We found that the additional requested group is significantly longer fixing time than the no requested group ($p << 0.05$).

V. Discussion

This section discusses some of our major findings from our case study.

In RQ1, we found out the frequently reported features across all types of HIB. Here, we did not consider the features that provided by reporters after initial submission. We have analyzed fixed bug reports in this case study. So, we may consider the most frequently reported features are more useful to fix HIB. However, at this stage, it is not clear whether they are really useful. This has lead to doing the second analysis.

In RQ2, we carefully examined developers and reporters each activity during bug fixing. Usually, developers request only for those features during bug fixing that are really helpful. Sometimes, they can not work without them. In this point of view, we can consider that additionally requested features are more useful to developers and have a higher impact on bug fixing. We found that developers made additional request for Test Cases, Code Examples, Steps to Reproduce, and Stack Traces are higher than others. Therefore, we can consider these four are key features for fixing HIB.

In RQ3, we tried to understand whether the request for additional features significantly affects on bug fixing time. Our significant test results suggest that the request for additional features significantly affects on bug fixing time. We tried to find out the reasons by analyzing each additionally requested features, requested time, and response time. We found that developers start working to fix the bug and find missing features in the bug reports that are essential to fix the bugs. Then, developers make the request for additional features to reporters and pause the bug fixing activities until the response of reporters. In many cases, reporters take a long time to response the request. This is the reason to affect the request for additional features significantly on bug fixing. Our case study findings will help to reduce the additional request during bug fixing. Findings from the case study will also help to develop automatic tools. If the reporters do not provide key features in the bug report then the tool may generate an automatic suggestion report to notify what additional features should be included in the bug reports. It also helps to write standard guideline on how to fill-up HIB report so that reporters can submit more accurate bug reports in the bug tracking system (BTS).

VI. Threats to Validity

For our case study we identified the following threats to validity.

We examined HIB reports of Apache Camel Project from Jira in our case study. There are some other BTSs such as Bugzilla. Every BTS follows their own convention and style to create and store bug reports. So, our findings may vary for the projects of others BTS.

We conducted a case study on HIB report based on MSR 2015 data showcase paper dataset. The data set contains limited number of HIB reports. So, our result may not be fully representative of their perspective.

We have analyzed HIB reports of open source projects. Sometimes, proprietary software differ from open source project. In this regard, our findings from this case study might not be applicable to the proprietary projects. We need to analyze more OSS projects as well as corporate projects to verify the generality of our findings.

VII. Related Work

Although we did not find any studies that directly research the topic of key features that play important role to fix high-impact bugs, we did find several studies that are related to our works. Bettenburg et al. [10] surveyed among 156 experienced developers and reporters of Apache, Eclipse and Mozilla to examine what features developers expect to see in the bug reports when fixing bugs. Based on the feedback of developers, they revealed the top 10 most important features that developers find useful during bug fixing. However, they did not investigate actual bug reports to find the usefulness of features. In our empirical case study, we used their survey result to understand how often reporters provide the 10 most important features in the bug reports and to how developers leverage them to fix the bugs.

Steven Davies et al. [8] conducted a case study on four open-source projects (Eclipse, Firefox, Apache HTTP, and Facebook API) to understand what information reporters provide in bug reports. They found that bug reports are neither complete nor accurate, and often do not provide all the information that developers find useful when fixing bugs. Furthermore, they also found that 12 percent of the total information are provided after initial submission of the bug reports. As a result, developers spend their valuable time to collect require information. However, in our case study, we especially focus on HIB reports to reveal the features that developers find useful to fixing the bugs. We also conduct a quantitative analysis to reveal how the request for additional features affects on bug fixing.

Shahed Zaman et al. [11] conducted a case study on Firefox project and studied how performance differ from security bugs in a software project. Authors found that security bugs are fixed and triaged much faster, but are reopened and tossed more frequently. Authors considered bug resolution time, triaging time, no. of reopen bug, developers experiences, no. of files changes etc. to make comparison between performance and security bugs. However, our goal is to find out features difference among different types of HIB.

Several studies that applied automatic techniques to select appropriate developers [16], localize bugs [17] and predict bug fixing effort [18]. All these approaches should benefit by our case study because, the performance of their model affect on the quality of the bug report.

VIII. Conclusion & Future Work

In this research, we conducted a case study on HIB reports of Apache Camel project to mine insightful information by analyzing reporters and developers activities. We manually analyzed each HIB report and extracted reported features. Then, we examined developers and reporters each activity during bug fixing. From our investigation, it is clear that in many cases, bug reports are not complete or accurate, and often do not provide features that developers find useful to fix the HIB. We found that for around 19% HIB report, developers collect useful features by making the additional request that causes significantly delay on bug fixing. We also found that Test Cases, Code Examples, Steps to Reproduce, and Stack Traces are the most additionally requested features. Our case study findings suggest that reporters should submit HIB report more accurately in order to promote the bug-fixing process. Our findings will be helpful to develop automatic tools recommending the bug reporters about the additional features that should be included in the HIB report. It will also help to formulate guidelines for reporters how to fill up bug report form more accurately. In the future, we plan to conduct a more comprehensive qualitative and quantitative study on different dimensional projects to generalize our case study findings. We also plan to extract features automatically from the bug reports using natural language techniques.

Acknowledgement

This work was supported by the JSPS Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem and the Grant-in-Aid for Young Scientists (B) (No. 16K16037).

References

[1] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07. ACM, 2007.

[2] Y. Kashiwa, H. Yoshiyuki, Y. Kukita, and M. Ohira, "A pilot study of diversity in high impact bugs," in *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, 2014.

[3] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: A study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. ACM, 2011.

[4] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. IEEE Press, 2012.

[5] T.-H. Chen, M. Nagappan, E. Shihab, and A. E. Hassan, "An empirical study of dormant bugs," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR '14. ACM, 2014.

[6] E. Shihab, A. Ihara, Y. Kamei, and W. Ibrahim, "Predicting re-opened bugs: A case study on the eclipse project," in *Proceedings of the 17th Working Conference on Reverse Engineering*, ser. WCRE '10. IEEE, 2010.

[7] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, and K. Matsumoto, "A dataset of high impact bugs: Manually-classified issue reports," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015.

[8] S. Davies and M. Roper, "What's in a bug report?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. ACM, 2014.

[9] J. Spolsky. Joel on software blog. FogBUGZ. [Online]. Available: http://www.joelonsoftware.com/news/fog0000000162.html

[10] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '08/FSE-16. ACM, 2008.

[11] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: A case study on firefox," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR, 2011.

[12] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. ACM, 2010.

[13] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR, 2014.

[14] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Proceedings of the 7th Working Conference on Mining Software Repositories*, ser. MSR '10. IEEE Press, 2010.

[15] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. IEEE Press, 2013.

[16] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. ACM, 2006.

[17] G. Canfora and L. Cerulo, "Fine grained indexing of software repositories to support impact analysis," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ser. MSR '06. ACM, 2006.

[18] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR, 2007.