Towards Investigating the Key Features to Report the High-impact Bugs

Md. Rejaul Karim*, Akinori Ihara*, Eunjong Choi*, Xin Yang† and Hajimu Iida* and Kenichi Matsumoto*
*Nara Institute of Science and Technology, Takayama, Ikoma, Nara 630-0192, Japan
†Osaka University.1-1 Yamadaoka, Suita, Osaka, Japan
Email: [rejaul.karim.qw4,akinori-i,choi,matumo]@is.naist.jp, xinyang@ist.osaka-u.ac.jp,iida@itc.naist.jp

***Abstract*—High impact bugs (Performance, Security, Breakage, Blocking, Surprise, and Dormant) should triage and fix faster than other bugs. In order to fix high-impact bugs, some information plays a greater role for some kind of high-impact bugs than others. However, in OSS projects, bug reporters do not give extra attention to submit high-impact bug reports and treat them equally. In addition, there is no standard guideline on how to write high-impact bugs more accurately. As a result, bug management process faces a number of difficulties such as assigning priority, selecting appropriate developers, and fixing bugs. Therefore, in this paper, we perform an in-depth empirical study on high-impact bug reports to mine insightful information making them better. Using high-impact bug reports of Apache Camel project, we conducted a case study. In details, we manually examined each high-impact bug reports and performed a qualitative and quantitative analysis. We observed that Test Cases, Code Examples, Steps to Reproduce, and Stack Traces are the most additionally requested features than others and the request for additional features significantly affects on bug fixing.**

I. Introduction

High impact bugs (Performance, Security, Breakage, Blocking, Surprise, and Dormant) should fix faster than other bugs. In order to fix the bugs, bug reports are the primary means and provide crucial information for developers (bug fixers). A bug report is a combination of structured and unstructured information. Among them, some information plays greater role for some kind of high-impact bugs than others. For instance, in case of performance bug, if reporters claim that the particular module of an application performs slowly. From this observed behavior, developers can understand the problems about the mentioned module of the application but it is also essential for developers to know about the reporter's expectation to fix the bug correctly. Otherwise, bug will be reopened in near future. On the other hand, in case of normal bugs, if the reporter claims that particular image does not render in a certain module. From this observed behavior, developers can understand the problems as well as what will make reporter's happy. Therefore, developers may fix and close the bug without clearly mentioned expected behavior in the bug reports. One case study found that different types of bugs (e.g. Performance and Security bugs) vary from each other [1]. They also found that security bugs are fixed and triaged faster, but are reopened and tossed more frequently. However, in OSS projects, bug reporters do not give extra attention to submit high-impact bugs. In addition, there is no standard guideline on how to write high-impact bugs more accurately. As a result, bug management process faces a number of difficulties such as assigning priority, selecting appropriate developers, and fixing bugs.

On the other hand, bug reporters face complexities to provide crucial information in the bug reports [2]. For instance, stack traces is very useful to localize bug [3]. However, bug reporters are not able to provide it for all bug reports, because, performance and functional bugs do not always produce it. We need to know how to make good quality bug reports by providing lesser information. Joel Spolsky once noted in his blog as follows:*"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"* [4]. In this sense, It is very important to make bug reports as simple as possible. But, it is not easy to understand for novice users and end users to know at least which information should be included in the bug reports for which type of bug reports at time of submission.

To the best of our knowledge, there is no case study on revealing key features set for each type of high-impact bugs. Therefore, we motivated to do an empirical case study on high-impact bug reports to investigate in details and to reveal insightful information about what makes a good high-impact bug report. As the first step of our research, we conducted a case study on high-impact bug reports of Apache Camel project [5]. We manually investigated each high-impact bug report and checked each reported information. Then, we analyzed each conversation between developers and reporters to understand when was information provided and how they affected on bug resolution. From our analysis, we have found that Test Cases, Code Example, and Steps to Reproduce are the most additionally requested features by developers during fixing the bugs. We also found that the request for additional features significantly affects on bug resolution. To achieve our goals, we answer the following research questions in this paper.

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

RQ2: Which features did developers request the most during bug fixing in each type of high-impact bugs?

RQ3: What are the effects of the request for additional

features on bug fixing time in high-impact bugs?

The main contributions of this research are:

1. We conducted a case study on high-impact bug reports of the Apache Camel project to reveal insightful information for fixing high-impact bugs. We observed that developers considered Test Cases, Code Examples, Steps to Reproduce, and Stack Traces are more useful than others.

2. We also perform a quantitative analysis to understand how significantly the request for the additional features affects bug fixing time. The p-values of Wilcoxon signed-rank test is 0.000063. It shows that the request for additional features significantly affect on bug fixing.

II. Background

In this section, we first briefly describe bug report and feature of a bug report. Next, we describe different types of high-impact bugs.

A. Bug Report

A bug report is a primary means by which users of a system are able to communicate a problem to the developers. When a bug identifies in a system, at first, reporters write a short summary on identified bug and then provides additional information to give a clear idea of the bug. Some of the fields are very straightforward and contain limit values such as version, component, and severity. A field, named description, is a free text and contain unstructured information.

B. Feature of Bug Report

A bug report is a combination of structured (e.g., version, severity, environment, reporter) and unstructured information (e.g., summary, Steps to Reproduce, Observed Behavior). Each piece of information that helps to describe a bug defines as a feature. Bettenburg et al. [2] surveyed among 156 experience developers 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. The table I shows the top 10 most important features to fix the bugs.

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

Name of Features	Short Description
Summary	Summary should be a short and precise description of the bug.A good summary helps the developers to understand the bug quickly and uniquely. It should explain the problem, not your suggested solution.
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

C. High-Impact Bug(HIB)

Software engineering researchers have introduced different high-impact bugs [6], [7], [8], [9], [10], [5] based on their impact on end-users and developers in software system. The following are the different types of high-impact bugs:

1) Surprise bugs: A surprise bug [7] 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 [9] 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). [11] showed that 33% of the reported bugs in Apache Software Foundation (ASF) projects were dormant bugs.
3) Blocking bugs: A blocking bug is a bug that blocks other bugs from being fixed [12]. It often happens because of a dependency relationship among software components. Since a blocking bug inhibit developers from fixing other dependent bugs, it can highly impact on developers task scheduling.
4) Security bugs: A security bug [13] can raise a serious problem which often impacts on uses of software products directly. Since Internet devices (e.g., smartphones) and their users are increasing every year, security issues of software products should be of interest to many people. In general, security bugs are supposed to be fixed as soon as possible.
5) Performance bugs: A performance bug [14] is defined as programming errors that cause significant performance degradation. The performance degradation contains poor user experience, lazy application responsiveness, lower system throughput, and needles waste of computational resources [15].
6) Breakage bugs: A breakage bug [5] 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 causes a problem which makes

usable functions in one version unusable after releasing newer versions.

III. Case Study Design

This section describes how we conducted the case study. First, we describe target dataset. Then, we describe the analysis procedure to address earlier mentioned research questions.

A. Target Dataset

High-impact bugs are more impactful on software processes, products, and end-users. It should be triaged and fixed earlier that others. Bug reports play an important in fixing the bugs. Therefore, our primary goal is to reveal insightful information to make good high-impact bug reports. As a target dataset, we selected high-impact bug reports from [5]. This dataset was created by manually reviewing four thousand issue reports in four open source projects (Ambari, Camel, Derby and Wicket). They were very careful about bias-free selection of bug reports and classified by multiple reviews. The table II shows the statistics of analyzed bug reports according to high-impact bugs.

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

Type of high-impact bug	No. of bug reports
Performance	51
Surprise	14
Breakage	39
Security	7
Dormant	69
Blocker	128

B. Analysis Procedure

To address our three research questions, we first filtered out high-impact bug reporters from Jira issue tracking system based on [5] for the Apache Camel project. Then, we manually extracted each reported features in the bug reports. After that, we carefully examined developers and reporters each activity and recorded features that were provided during bug fixing. Finally, we performed qualitative and quantitative analysis to address our research questions. The figure 1 describes overall analysis procedure.

1) Qualitative Analysis: We have conducted a qualitative analysis on high-impact reports to achieve our goals. We manually analyzed each high-impact reports. Some fields of a bug report, such as the severity or version, can only take a limited number of values. Features extraction from these fields are relatively straightforward manner using automated techniques. However, there are a number of other features that are desirable in a bug report, which are not contained in separate fields, such as Steps to Reproduce, Expected Behavior. Unfortunately, in general, BTSs does not have specific support for any of these features, and they are usually provided in description or in a file attachment. All of these are unstructured natural language plain text. In some cases, they mention steps to reproduce key, expected behavior key words

Fig. 1. Overall HIB reports analysis procedure

to describe the features in the description section. However, most of the cases, they do not mention phrases Steps to Reproduce. Therefore, we examined manually each of the high-impact bug reports from the developer's perspective and recorded each of the reported features in the bug reports. Sometimes, bug reporters do not provide some features at initial submission.They provide it based on the request of developers. Therefore, we also examined each conversation between developers and reporters to understand features that was requested to provide by developers.

2) Quantitative Analysis: In this case study, we also conducted a quantitative analysis to understand whether the request for additional features significantly affects on the bug fixing. In our quantitative analysis, we first divided bug reports into two groups. One group is for additional features request (Res) and another group is for no additional features request (non-Res). Then extract bug report creation date time and bug resolution date time. After that, we calculated bug fixing time for both group. Finally, we applied wilcoxon signed-rank test to know whether the effect of the request for additional features on bug fixing is significant.

IV. Analysis Result

This section presents the results for the research question. To get the answers of our research questions first, we carefully examined each reported features in the bug reports. Second, we examined developers and bug reporters each activity to understand the features that developers find useful during bug fixing. Finally, we analyzed the effect of additional request for features on bug fixing.

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

The results for the research question 1 are summarized in the figure 2. We see that the average percentages of reported features in Performance, security, Breakage, Surprise, Dormant, and Blocker bug reports are 33%, 33%, 32%, 29%, 27%, and 31% respectively. Whether that is sufficient or not cannot be discussed at this point, since further research on the topic is needed. We also see that the highest percentages of reported features is Observe Behavior among all type of high-impact bugs and Screenshot and Error Report are provided in a very few percentages bug reports. We can see

Fig. 2. Observed features in terms of high-impact Bugs

that reporters provided Expected Behavior comparatively in higher percentages in Performance and Security bug reports than others. Other features vary among different types of high-impact bugs.Therefore, at this stage, it is not clear which features have a comparatively higher impact on fixing high-impact bugs than others have. This has lead to the examination of developers and reporters activities during bug fixing for the RQ2

RQ2: Which features did developers request the most during bug fixing in each type of high-impact bugs?

To answer this question, we examined each activity and conversation between developers and reporters during bug fixing. We are summarized the analyzed result in the table III. We found that developers request for additional features on average for 20% bug reports. Among the 20% bug reports, in case of Performance bug, 54% request for Test Cases, 23% request for Code Example, 15% request for Steps to Reproduce, and 8% request for Stack Traces. In our examination, we observed that developers did not make any request to provide additional feature for the blocker bug reports. It does not mean developers do not make additional request for the blocker bug reports. Actually, our analyzed dataset contains a small number of blocker bug reports. We need to analyzed more blocker bug reports to get the actual picture. However, this result can be a good source to understand the useful features that play an important role to fix the high-impact bugs because developers request only for those features that are really helpful for them to localizing and fixing the bugs correctly.

RQ3: What are the effects of the request for additional features on bug fixing time in high-impact bugs?

To answer this question, we first divided bug reports into two groups. One group is for additional features request (Res) and another group is for no additional features request (No-Res). Then, we extracted the bug report creation date time and the bug resolution date time. After that, we calculated bug fixing time for both groups. The figure 3 shows the distribution of bug fixing time among additional features request and no-request groups.

We observed that the median bug fixing time of additional features requested group is comparatively higher than the

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%

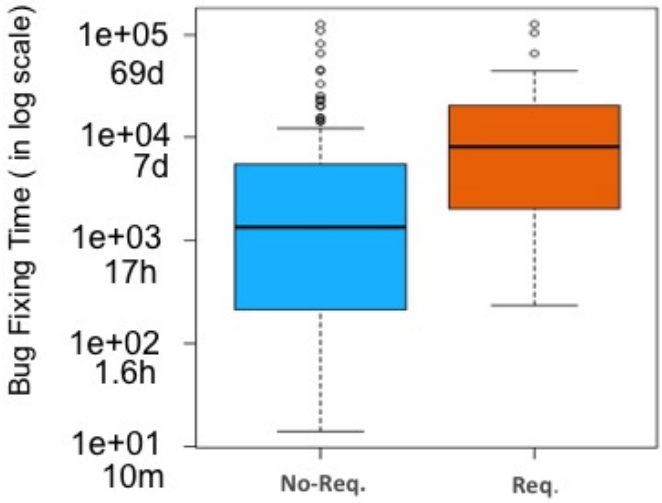

Fig. 3. Distribution of bug fixing time between additional features request and no-request groups

no-request. Therefore, we conducted a Wilcoxon signed-rank test, to know whether the effect for the request of additional features on bug fixing time is significant. In the test, our null hypothesis H_0 and alternative hypothesis Ha are as follows:

H_0: There is no statistical difference in bug fixing time between the additional features request and no-request groups.

Ha :The bug fixing time of the additional feature request group is statistically higher than the no-requested group.

The null hypothesis H_0 can be rejected and the alternative hypothesis Ha will be accepted when p-value of the statistical test is below the standard significant value, i.e., ¡ 0.05. In this study, the Wilcoxon signed-rank test, a non-parametric statistical hypothesis test, was used. This test was selected because the distribution of bug fixing time does not follow a normal distribution. As an outcome of the significant test, we got p-values is 0.000063. It indicates that the request for additional features significantly affects on bug resolution.

V. DISCUSSION

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

In our first analysis (RQ1), we find out the frequently reported features across all types of high-impact bugs. 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 high-impact bugs than others are. However, at this stage, it is not clear whether they are really useful or not. This has lead to doing the second analysis.

In our second analysis (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 high-impact bugs.

In our third analysis (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 that are essential to fix. 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 high-impact bug 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 high-impact bug reports of Apache Camel Project from Jira in our case study. There are some other bug tracking systems (BTS) such as Bugzilla. Every BTS follows their own convention and style to create and store bug reports. So, our finds may vary for the projects of others BTS.

We conducted a case study on high-impact bug reports and normal bug reports of MSR 2015 data showcase paper dataset. The data set contains limited number of high-impact bug reports. So, our result may not be fully representative of their perspective.

We have analyzed high-impact bug reports as well as other bug 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 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. [2] surveyed among 156 experience 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 find them useful to fix the bugs.

Steven Davies et al. [16] 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 high-impact bug reports to reveal the features that developers find most useful during fixing by analysis reported features in the bug reports and conversation between developers and reporters during bug fixing. We also conducted a quantitative analysis to reveal how the request for additional features affects on bug fixing.

Shahed Zaman et al. [1] 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 concern about how to file a high-impact bug reports more accurately.

Several studies used bug reports to automatically assign developers to bug reports [17], assign locations to bug reports [18], and predict effort for bug reports [19]. 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 high-impact bug reports of Apache Camel project to mine insightful information by analyzing reporters and developers activities. We manually analyzed each high-impact bug reports and extract 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 would wish to find, or could use to fix bugs. We found that around 20% high-impact bug reports collect require features by making the additional request that causes significantly delay on bug fixing. We also found that developers make the additional request for Test Cases, Code Examples, Steps to Reproduce, and Stack Traces comparatively higher that others. Our case study findings suggest that reporters should submit high-impact bug reports 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 high-impact bug reports. It will also help to formulate guidelines for reporters how to fill up bug report form more accurately. In future, we will conduct a more comprehensive qualitative and quantitative study on different dimensional projects to generalize our case study findings. We also have a plan to work on proposing a novel approach to extract features automatically from the bug report using natural language techniques.

Acknowledgment

The authors would like to thank...

References

[1] 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 '11. ACM, 2011, pp. 93–102.

[2] 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, pp. 308–318.

[3] A. Schroter, N. Bettenburg, and R. Premraj, "Do stack traces help developers fix bugs?" in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, ser. MSR '10. IEEE, 2010, pp. 27–30.

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

[5] 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.

[6] 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.

[7] 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, pp. 300–310.

[8] 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, pp. 1074–1083.

[9] 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, pp. 82–91.

[10] 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, pp. 249 – 258.

[11] 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, pp. 45–54.

[12] 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 '14. ACM, 2014, pp. 72–81.

[13] 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.

[14] 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, pp. 237–246.

[15] I. Molyneaux, *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*, 1st ed. O'Reilly Media, Inc., 2009.

[16] 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, pp. 26:1–26:10.

[17] 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, pp. 361–370.

[18] 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, pp. 105–111.

[19] 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 '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1–.