Vol.2016-SE-192 No.9
Vol.2016-EMB-41 No.9
2016/6/3

# Improving the High-Impact Bug Reports: A Case Study of Apache Projects

Md. Rejaul Karim[1,a) ] Akinori Ihara[1,b)] Xin Yang[1,c)] Eunjong Choi[1,d)]
Hajimu Iida[1,e)] Kenichi Matsumoto[1,f)]

***Abstract:***
Good quality bug reports are the primary means for developers to fix the bugs. However, the quality of the bug reports depends on the contents that help developers to resolve the bugs. This research aims to investigate what and how information is provided in the bug reports. Using high-impact bug reports of Apache Camel project, we conducted a case study. In details, we manually investigated high-impact bug reports to identify frequently reported features and how they affect on the bug fixing time. As a result, we found out frequently reported features set for each type of high-impact bugs. Moreover, no strong relationship exists between provided features and the bug fixing time.

***Keywords:*** Bug Report, Bug Tracking System, Open Source, Software Repository Mining

## 1. Introduction

One of the key activities in the software development process is fixing bugs reported by developers, testers, and end users [23]. To fix the bugs, developers typically rely on the information that is contained in the bug reports [8]. Good quality bug reports help the developers to fix the bugs [2]. However, the reporters do not always provide adequate information in the bug reports.

Davies and Roper found that bug reports are neither complete nor accurate, and often do not provide all the information that developers find useful when fixing bugs. They also found that 12 percent of the total information are provided after initial submission of the bug reports [5]. As a result, developers spend their valuable time to collect require information. Furthermore, it revealed that it is difficult to automatically extract relevant information from bug reports..

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 [15]. 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"* [18]. 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 which information are important for which type of bug reports at time of submission. We need to find out important information set according to high-impact bugs that developers find useful during bug fixing.

The main objectives of this study are to find out frequently reported information for each category of high-impact bugs to understand important information that developers find useful during bug fixing. We expect that our findings help to develop a tool suggesting the bug reporters how to submit more accurate high-impact bug reports in the bug tracking system (BTS). It also helps to write standard guideline how to fill-up bug reports. Consequently, quality of high-impact bug reports will be improved.

We have done a case study on high-impact bug reports of Apache Camel project [13]. 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 different information set for each type of high-impact bugs based on how often appears in the bug reports. We also found interesting relationship between bug fixing time and no. of reported features in each type of high-impact bugs.

To provide evidence to show the importance of individual features for high-impact bugs, we answer the following research questions in this paper.

RQ1: Which features are important to fix high-impact bugs for each category?

RQ2: Is there any relationship between bug fixing times with no. of reported features?

The structure of this paper is as follows. In Section 2, we explain on bug report, high-impact bug report, and important fea-

[1] Nara Institute of Science and Technology, Takayama, Ikoma, Nara 630-0192, Japan
[a)] rejaul.karim.qw4@is.naist.jp
[b)] akinori-i@is.naist.jp
[c)] kin-y@is.naist.jp
[d)] choi@is.naist.jp
[e)] iida@itc.naist.jp
[f)] matumoto@is.naist.jp

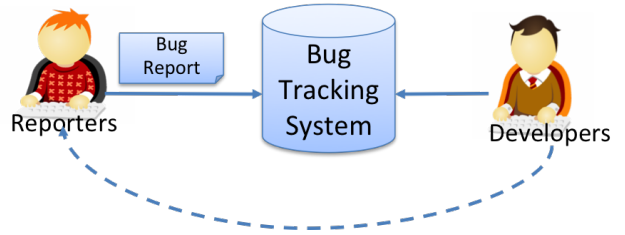


**Fig. 1** Bug reporting system for OSS projects.

tures of bug report. In Section 3, we present the motivation of this study with example of bug reports. Next, we describe target dataset and analysis procedure in section 4. In Section 5, we present results of the case study. In Section 6, we explain our major findings and how our case study can contribute in the bug management process. In Section 7 and 8, we discuss the threats to validity of our case study and related work, respectively. Finally, we conclude and give future direction of our research in Section 9.

## 2. Background

Bug reporters (e.g. software developers, testers, and end users) identify bugs in software projects.They submit the identified bugs into the bug tracking system (BTS). Figure 1 shows the conventional bug reporting system for Open Source Software (OSS) projects. Then, each bug report is assigned to the appropriate developer to fix the bug. However, in some cases, bug reporters do not provide information that are very useful for developers. Therefore, developers request to reporters providing additional information. Then, bug reporters provide requested information in the comment section of bug reports. In this section, we explain on bug report, feature and high-impact bug to understand this study.

### 2.1 Bug Report

BTS has an interface with some fields to create a new bug report. 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 data. In the bug report form, there are no separate fields for Steps to Reproduce, Stack Traces, and some others. Therefore, reporters provide these sorts of information in the description field. In OSS projects, bug reporting is a voluntary work. Sometimes, they do not following guidelines of submitting bug reports. For example, they do not mention the key word ”Expected Behavior” for describing their expectation from the patches or the pieces of code. As a result, developers find difficulties to understand the bug properly.

### 2.2 Feature

A bug report usually consists of a number of fields. Each field may contains one or more values. Each piece of information that helps to describe a bug defines as a feature. Therefore, a bug report contains many features such as components, expected behavior. Among them, developers consider some are more important than other features to fix the bugs. Bettenburg et al [2] surveyed 156 developers of Apache, Eclipse and Mozilla to examine what information developers want when fixing bugs. The survey result revealed that developers regard the ten features (shown in Table 1) as important in fixing bugs.

**Table 1** List of top 10 important features

| 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 expected to happen, usually contrasted with Observed behaviour |
| 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 |

### 2.3 High-Impact Bug

Software engineering researchers have introduced different high-impact bugs [4], [9], [13], [16], [17], [23] 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 [17] 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). As a result of a case study of a proprietary, telephony system which has been developed for 30 years, [17] showed that the number of surprise bugs were very small (found in 2% of all files) and that the co-changed files and the amount of time between the latest pre-release date for changes and the release date can be good indicators of predicting surprise bugs.

( 2 ) Dormant bugs: A dormant bug [4] 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). [19] showed that 33% of the reported bugs in Apache Software Foundation (ASF) projects were dormant bugs and were fixed faster than non-dormant bugs. It indicates that dormant bugs also affect developers workflow in fixing assigned bugs in order to give first priority to fix the dormant bugs.

( 3 ) Blocking bugs: A blocking bug is a bug that blocks other bugs from being fixed [20]. 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 since a blocking bug takes more time to be fixed [20] (i.e., a fixer needs more time to fix a blocking bug and other developers need to wait for being fixed to fix the dependent bugs).

(4) Security bugs: A security bug [6] 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 [12] 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 [11]. [12] showed that a performance bug needs more time to be fixed than a non-performance bug. Therefore, performance bugs can affect users for a long time.

(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. Though it is well-known as regression, a breakage bug mainly focuses on regression in functionalities. A breakage bug causes a problem which makes usable functions in one version unusable after releasing newer versions.

## 3. Motivation of this study

We found in Camel Project, eight percent of total bug reports are marked as Invalid, two percent of total bug reports are marked as unresolved, a number of bug reports marked as Open. We also found that bug fixing time vary among the bugs. The figure 2 shows an example of bug report extracted from Apache Camel Bug ID Camel-3540.

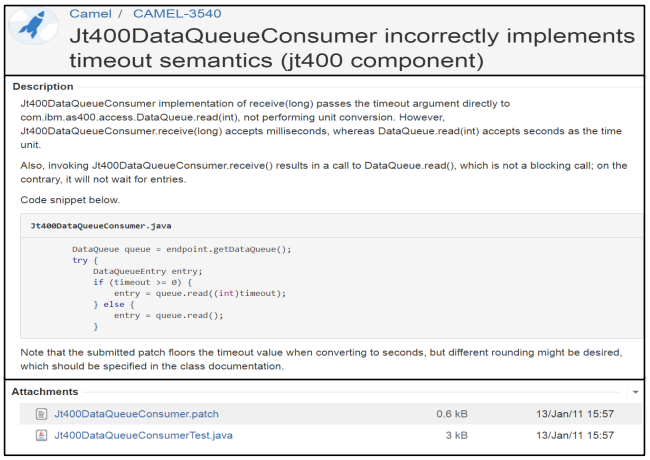


**Fig. 2** Description of Apache Camel Bug ID-Camel-3540

It is a performance bug. This bug report is detailed, and contains clear information about bug. Reporter provided code snippets to describe where the exact problem happened. Reporter also attached test cases in the report for developer to test after fixed. This type of bug report is easy to understand for developers. Developer fixed the bug on the same day of reporting.

The figure 3 shows an example of bug report extracted from Apache Camel Bug ID Camel-5860. It is a Surprise bug.

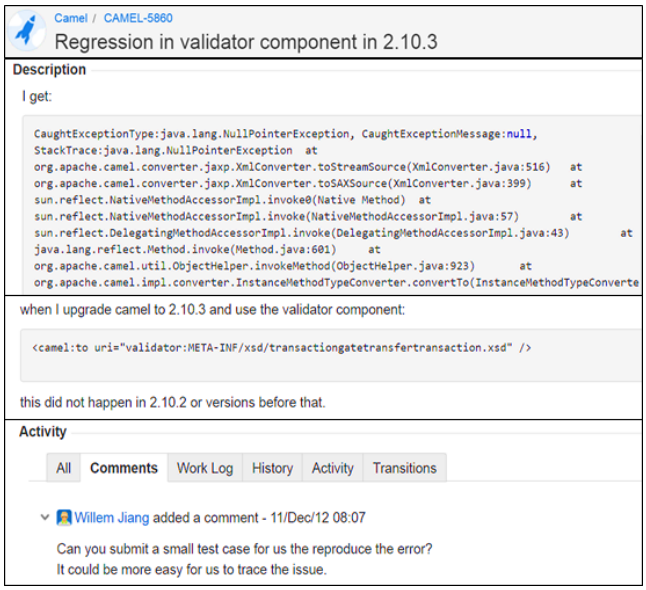


**Fig. 3** Description of Apache Camel Bug ID-Camel-5860

This bug report was created in December 10, 2012. Reporter provided Stack Traces and very minimal information about the bug. Developer could not understand the exact problem clearly. He also could not understand how to reproduce the bug. So, one day later, developer requested to provide additional information *(Can you submit a small test case for us the reproduce the error?)* about the bug. Two months 17 days (on February 27, 2013) later, reporter responded and provided Steps to Reproduce. Finally, developer fixed the bug after 8 days. Actually, bug fixing time depends on many factors such as complexities of bug, importance of bug, and developer's experience. However, It also depends on the quality of bug reports.

Among the bug reports, high-impact bug reports are more impactful on software system. For instance, a security bug must be fixed faster than other bugs. A performance bug can directly decrease the satisfaction of users on products. For these reasons, developers try to fix the these sort of bugs as early as possible. Time delay to fix high-impact bugs is very costly. Therefore, In case of high-impact bug, good quality bug reports are more important than others.

There are some researches on identifying, classifying, and predicting high-impact bugs. To the best of our knowledge, there is no case study on revealing important features set for each type of high-impact bug. Therefore, we motivated to do research on how to improve the quality of high-impact bug reports. As the first step of our research, we try to reveal frequently reported features by analyzing historical bug reports. It helps us to understand the important features set according to high-impact bugs.

## 4. Case Study Design

In this section, we describe our case study on high-impact bug

reports that aims to examine how often reporters provide each of the top 10 important features. It helps to understand the important features and how affect these features on bug resolution according to high-impact bugs. First, we introduce our dataset and then explains why we choose the top 10 features for our analysis. Finally, we explain our analysis procedure of high-impact bug reports.

### 4.1 Analyzed Dataset

We have analyzed high-impact bug reports of Apache Camel Project from JIRA, BTS. As a target dataset, we selected a Dataset of high impact bugs [13]. 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 bug reports selection and classification by multiple reviews. The table 2 shows the statistics of analyzed bug reports according to high-impact bugs.

**Table 2** 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 |

### 4.2 Selected Features

Bettenburg et al. [2] surveyed among 156 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 that steps to reproduce is most important feature. Davies et al. [5] examined that observed behavior is the most important feature based on analysis how often reporters provide every feature in the bug reports. So, we have selected the top ten features mentioned in the section 2.2 for our analysis to examine important features set according to high-impact bug from the developer perspective.

### 4.3 Analysis procedures

Some fields of a bug report, such as the severity or version, can only take a limited number of values. Features can be extracted from these fields in a 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 have no specific support for any of these features, and they are usually provided in description or comments. All of these are unstructured plain text, or as generic attachments. In some cases, they mention steps to reproduce key, expected behavior key words to describe the features in the description section. However, most of the cases, they do not mention. Therefore, we examined manually each of the high-impact bug reports from the developer's perspective and recorded each of the reported features from 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

## 5. Analysis Results

In this section, we address the answers of the RQs mentioned in Section 1. To answer RQs, at first, we examine frequently reported features set according to high-impact bugs by analyzing historical bug reports. Then, we try to explore relationship between no. of reported features with bug resolution time.

**RQ1: Which features are important to fix high-impact bugs for each category?**

To answer this question, we examined each high-impact bug reports. We observed how often bug reporters provided each top 10 features in the bug reports at initial submission. Then, we examined the features that was requested to provide by developers at comment section. After that, we combine the features reported in both initial submission and comment section to calculate how often reporters provide each features in each bug reports. Our observations are summarized in the figure 4 as a bar chart according to high-impact bugs. The x-axis of the bar chart represents the features and the y-axis represents the total number.

The figure 4(A) shows the number of reported features for security bug. The mean of the reported features is 45.25. The features, Steps to reproduce, Observed behavior, Expected behavior, Test cases, and Code examples are found in the bug reports more than mean. So, the most frequently reported features set for security bug is Observed Behavior, Expected Behavior,Test Cases, and Steps to reproduce

The figure 4(B) shows the number of reported features for Surprise bug. The mean of the reported features is 34.55. The features, Observed Behavior, Expected Behavior, Test Cases, and Code Examples are found in the bug reports more than mean. So, the most frequently reported features set for breakage is Observed Behavior, Code Examples, Expected Behavior, and Test Cases

The figure 4(C) shows the number of reported features for performance bug. The mean of the reported features is 35.95. The features, Observed Behavior, Expected Behavior, Test Cases, and Code Examples are found in the bug reports more than mean. So, the most frequently reported features set for performance bug is Observed Behavior, Expected Behavior,Test Cases, and Code Examples

The figure 4(D) shows the number of reported features for Dormant bug. The mean of the reported features is 37.86. The features, Steps to reproduce, Observed Behavior, Expected Behavior, Test Cases, and Code Examples are found in the bug reports more than mean. So, the most frequently reported features set for breakage bug is Observed Behavior, Code Examples, Steps to reproduce, and Test Cases

The figure 4(E) shows the number of reported features for Breakage bug. The mean of the reported features is 25.64. The features, Observed Behavior, Stack Traces, and Code Examples are found in the bug reports more than mean. So, the most frequently reported features set for breakage bug is Observed Behavior, Code Examples, Stack Traces, and Steps to reproduce

The figure 4(F) shows the number of reported features for

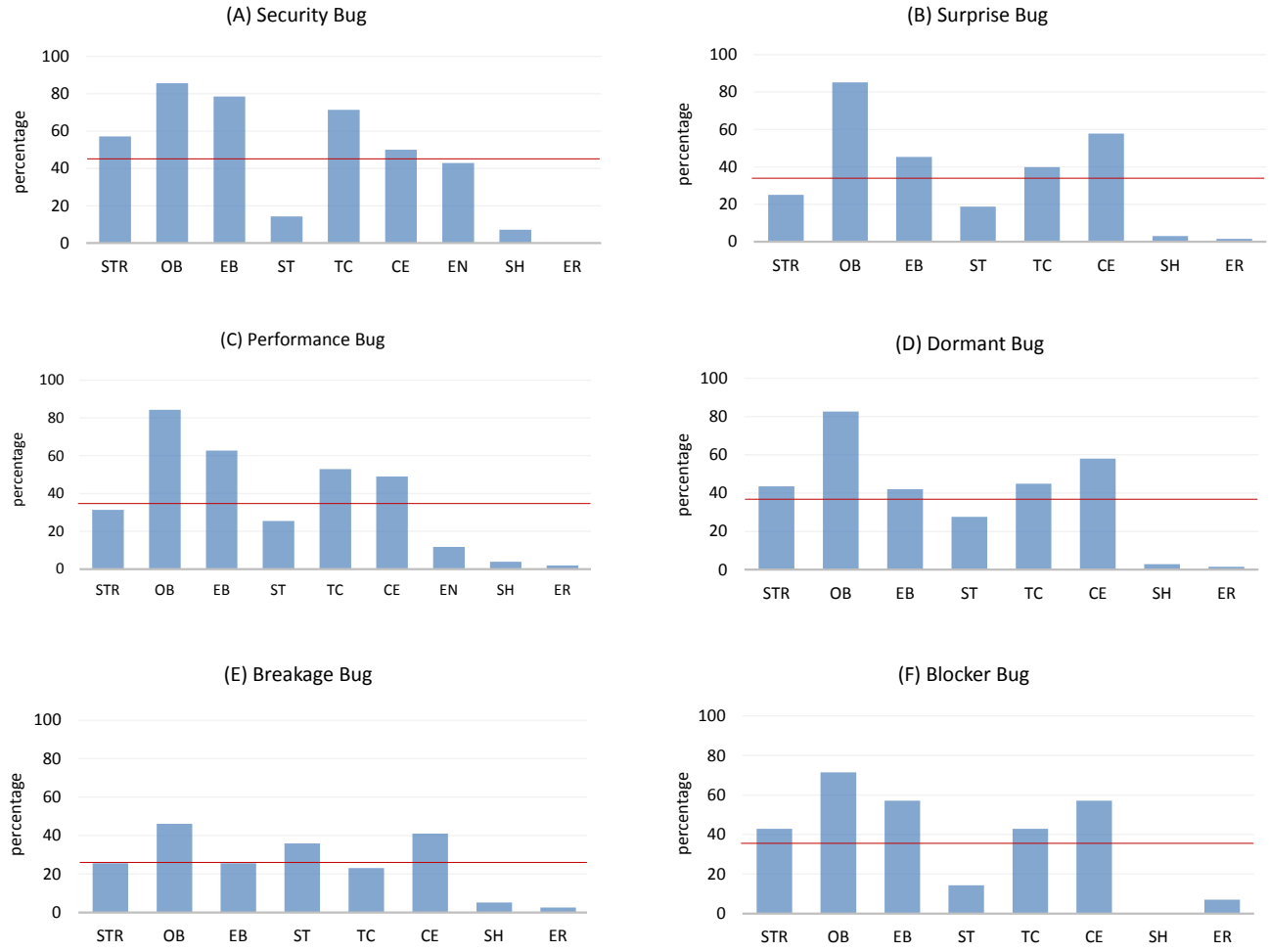


**Fig. 4** Observed features in terms of high-impact Bugs

Blocker bug. The mean of the reported features is 36.59. The features, Steps to reproduce, Observed Behavior, Expected Behavior, Test Cases, and Code Examples are found in the bug reports more than mean. So, the most frequently reported features set is Observed Behavior, Code Examples, Test Cases, and Steps to Reproduce

**RQ2: Is there any relationship between bug fixing times with no. of reported features?**

We set up this RQ to identify whether reported features affect on bug fixing time or not. To answer this RQ, we calculated the required time to fix a bug by subtracting resolution time from reporting time. Then, we calculated the average bug fixing time, average bug fixing time when no. of reported features greater than or equal fours, and average bug fixing time when no. of reported features less than fours. We summarize our findings in table 3.

**Table 3** Average bug resolution time in terms of high-impact Bugs

| Type of high-impact bug | Overall time (in minutes) | ≥ 4 features (in minutes) | <4 features (in minutes) |
|---|---|---|---|
| Performance | 184.52 | 70.56 | 270.98 |
| Surprise | 142.82 | 104.66 | 174.43 |
| Breakage | 250.78 | 140.64 | 316.86 |
| Security | 59.77 | 35.56 | 92.04 |
| Dormant | 110.21 | 183.12 | 61.92 |
| Blocker | 109.91 | 156.61 | 60.22 |

The table 3 shows that the average bug fixing time are minimum for Performance, Surprise, Breakage, and Security when the no. of reported features greater than or equal fours. In case of Dormant and Blocker, average bug fixing time is higher the when no. of reported features greater than or equal fours.

## 6. Discussions

In this section, we explain our major findings and how our case study can contribute in the bug management process:

In our first analysis (RQ1), we find out the frequently reported features set across all types of high-impact bugs. During bug fixing, developers usually ask reporters to provide more features that find useful to fix the bug. To calculate frequently reported features, we consider both initially submitted features and features that request in the comment section during bug fixing. Therefore, our findings suggest that frequently reported features set may be the important features set across all type of high-impact bugs.

In our first analysis, we find that 5 features above the mean value for Security, Dormant, and Blocker bugs. 4 features are above the mean value for Surprise and Performance bugs. In case of Breakage, 3 features are above than the mean value. So, we suggest important features set for each type of high-impact bugs that are the combination of 4 features. For this reason, we try to

understand if the reporters provide at least 4 features in the bug reports whether it affects on bug resolution.

In our second analysis (RQ2), we find the average bug-fixing time for performance, surprise, Breakage, and Security are lower when the number of reported features greater than or equal fours. In case of Dormant and Blocker, we find the average bug fixing time is higher when no. of reported features greater than or equal fours.

From the analysis result, we can conclude that if bug reporters provide at least four features among the top 10 features in the bug report then bug-fixing time may reduce. Because, in that case, the bug report is easy to understand that helps to localize and to assign appropriate developer to fix the bug.

Our findings from the case study help to develop automatic tools. If the reporters do not provide important features in the bug report then tool may generate automatic suggestion report to notify what additional features should be included in the bug reports. It also helps to write standard guideline how to fill-up high-impact bug report so that reporters can submit more accurate bug reports in the bug tracking system (BTS). In this way, our case study can contribute to improve bug management process.

## 7. Threads 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. Many open source projects use Jira as BTS. So, our finds may not be applicable for others projects.

We used dataset of high-impact bug for our analysis. The data set contains limited number of high-impact bug reports. So, our result may not be fully representative of their perspective.

We investigated frequently reported features set according to high-impact bugs. For example, Observed Behavior, Expected Behavior, Test Cases, and Steps to reproduce considers the most useful features for security bug based observing frequently appear these features in the bug reports. But, actual developers may find different features set for each type of high-impact bugs.

We analyzed high-impact bug reports of open source projects. But our findings from this case study might not be applicable to any corporate projects. We need to analyze OSS projects as well as corporate projects to verify the generality of our findings.

## 8. Related Works

Hooimeijer and Weimer built a descriptive model to predict whether a bug report is triaged within a given amount of time [7]. They assumed that description readability is a good indicator for the quality of a bug report. In contrast, our notion of quality is based the readability as well as contained of the description of a bug report. Developers find the most useful information in description. Therefore, measuring only readability does not enough to evaluate the quality of bug report.

Buttenburg et al. [2] investigated the quality of bug reports from the perspective of developers.They surveyed among experience developers and bug reporters to understand important features for bug reports. From the surveyed results, they found information mismatch between what developers consider most helpful and what users provide. They also developed CUEZILLA to measure the quality of bug reports from very poor to very good on five-point Likert scale. However, based on the surveyed results, they point out top 10 important features for developers to fix the bugs. In this case study, we analyzed carefully each high-impact bug reports to find out how often reporters provides these top 10 important features in the bug reports. We also analyzed the relation between no. of reported features and bug resolution to understand how affect reported features on bug fixing time.This result will help to build recommendation system to suggest how to report bug more accurately at initial submission.

Rocha et al. [14] done an empirical study on recommendations of similar bugs and developed a tool named NextBug to recommend similar bug reports to developers. They used description to identify similar bug reports. Therefore, the accuracy of NextBug depends on the content of description. Our case study can be a first step to improve the quality of description of bug reports.

In order to inform the design of new bug reporting tools, Ko et al. [10] conducted a linguistic analysis of the titles and description of bug reports. They observed a large degree of regularity and a substantial number of references to visible software entities, physical devices, or user actions. Their results suggest that future bug tracking systems should collect data in a more structured way.

Several studies used bug reports to automatically assign developers to bug reports [1], assign locations to bug reports [3], and predict effort for bug reports [21]. All these approaches should benefit by our case study because, the performance of their model affect on the quality of the bug report.

Xia et al. [22] proposed a new and accurate method named DevRec for the developer recommendation problem. They use summary and description to generate terms and topics to build prediction model for DevRec. Then, they apply K-nearest neighbors algorithm on the topics of new bug reports and all historical bug reports to find the similar bug reports. Gegick et al. [6] used IR based technique to identify Security. Ohira et al. [13] classified different type of high-impact bug based on the content of bug reports. All these approaches rely on good quality bug reports. Our case study will help tools developers to suggest how to report bug more accurately in order to improve the quality bug reports.

## 9. Conclusions and Future Works

In this paper we conducted a case study on Apache Camel project to investigate frequently reported features according to high-impact bugs. We manually checked each bug reports and analyzed reported features from the developers perspective.We found that the highest number of features are reported for security bug and the lowest number of features reported for Breakage. Observed behavior reported in the highest number of bug reports across all types of high-impact bugs. Screenshots and Error Reports are found in a very few bug reports.We also found that similar features set in Surprise, Performance and Dormant, Blocker bug based on reported features. Our findings will help to develop tools recommending the bug reporters about the additional fea-

tures 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. Our future works are as follows:

**Study more Datasets:** Currently, We analyzed only high-impact bug reports of Apache Camel project. We plan to examine high-impact bug reports of different application domain projects that will help to mitigate generalization threads of our study.

**Survey among experience developers:** We found different features set according to high-impact bugs. We plan to survey among experience developers to validate our findings. It will help to build more useful features set according to high-impact bugs.

**Develop automatic recommendation tool:** Some bug reporters such as novice users, end users do not have proper knowledge about features that should be included in the bug reports. Automatic suggestion tool might be helpful for them to improve the quality of bug reports. We plan to develop an automatic recommendation tool to suggest how to report high-impact bugs more accurately.

## 10. Acknowledgements

This work has been supported by Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem. Also, part of this research was conducted under the Japan Society for the Promotion of Science, Grant-in-Aid for Young Scientists (B:16K16037).

### References

[1] Anvik, J., Hiew, L. and Murphy, G. C.: Who Should Fix This Bug?, *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, ACM, pp. 361–370 (2006).

[2] Bettenburg, N., Just, S., Schroter, A., Weiss, C., Premraj, R. and Zimmermann, T.: What Makes a Good Bug Report?, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, ACM, pp. 308–318 (2008).

[3] Canfora, G. and Cerulo, L.: Fine Grained Indexing of Software Repositories to Support Impact Analysis, *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, ACM, pp. 105–111 (2006).

[4] Chen, T.-H., Nagappan, M., Shihab, E. and Hassan, A. E.: An Empirical Study of Dormant Bugs, *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14, ACM, pp. 82–91 (2014).

[5] Davies, S. and Roper, M.: What's in a Bug Report?, *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, ACM, pp. 26:1–26:10 (2014).

[6] Gegick, M., Rotella, P. and Xie, T.: Identifying Security Bug Reports via Text Mining: An Industrial Case Study, *Proceedings of the 7th Working Conference on Mining Software Repositories*, MSR '10, IEEE Press (2010).

[7] Hooimeijer, P. and Weimer, W.: Modeling Bug Report Quality, *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, ACM, pp. 34–43 (2007).

[8] Just, S., Premraj, R. and Zimmermann, T.: Towards the Next Generation of Bug Tracking Systems, *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC*, ICSE '09 (2008).

[9] Kashiwa, Y., Yoshiyuki, H., Kukita, Y. and Ohira, M.: A Pilot Study of Diversity in High Impact Bugs, *Proceedings of the 30th International Conference on Software Maintenance and Evolution* (2014).

[10] Ko, A. J., Myers, B. A. and Chau, D. H.: A Linguistic Analysis of How People Describe Software Problems, *Proceedings of the Visual Languages and Human-Centric Computing*, VLHCC '06, IEEE Computer Society, pp. 127–134 (2006).

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

[12] Nistor, A., Jiang, T. and Tan, L.: Discovering, Reporting, and Fixing Performance Bugs, *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, IEEE Press, pp. 237–246 (2013).

[13] Ohira, M., Kashiwa, Y., Yamatani, Y., Yoshiyuki, H., Maeda, Y., Limsettho, N., Fujino, K., Hata, H., Ihara, A. and Matsumoto, K.: A Dataset of High Impact Bugs: Manually-classified Issue Reports, *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, IEEE Press (2015).

[14] Rocha, H., Valente, M., Marques-Neto, H. and Murphy, G. C.: An Empirical Study on Recommendations of Similar Bugs, *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, SANER '16, IEEE Press, pp. 46–56 (2016).

[15] Schroter, A., Bettenburg, N. and Premraj, R.: Do Stack Traces Help Developers Fix Bugs?, *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*, MSR '10, IEEE, pp. 27–30 (2010).

[16] Shihab, E., Ihara, A., Kamei, Y. and Ibrahim, W.: Predicting Reopened Bugs: A Case Study on the Eclipse Project, *Proceedings of the 17th Working Conference on Reverse Engineering*, WCRE '10, IEEE, pp. 249 – 258 (2010).

[17] Shihab, E., Mockus, A., Kamei, Y., Adams, B. and Hassan, A. E.: High-impact Defects: A Study of Breakage and Surprise Defects, *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, ACM, pp. 300–310 (2011).

[18] Spolsky, J.: Joel on Software blog, FogBUGZ (online), available from ⟨http://www.joelonsoftware.com/news/fog0000000162.html⟩ (accessed 2000-011-22).

[19] Sun, C., Lo, D., Wang, X., Jiang, J. and Khoo, S.-C.: A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval, *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, ACM, pp. 45–54 (2010).

[20] Valdivia Garcia, H. and Shihab, E.: Characterizing and Predicting Blocking Bugs in Open Source Projects, *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14, ACM, pp. 72–81 (2014).

[21] Weiss, C., Premraj, R., Zimmermann, T. and Zeller, A.: How Long Will It Take to Fix This Bug?, *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, Washington, DC, USA, IEEE Computer Society, pp. 1– (2007).

[22] Xia, X., Lo, D., Wang, X. and Zhou, B.: Accurate Developer Recommendation for Bug Resolution, *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*, WCRW '13, Koblenz, Germany, IEEE, pp. 72–81 (2013).

[23] Zimmermann, T., Nagappan, N., Guo, P. J. and Murphy, B.: Characterizing and Predicting Which Bugs Get Reopened, *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, IEEE Press, pp. 1074–1083 (2012).