2015 12th Working Conference on Mining Software Repositories

A Dataset of High Impact Bugs: Manually-Classified Issue Reports

Masao Ohira*, Yutaro Kashiwa*, Yosuke Yamatani*, Hayato Yoshiyuki*, Yoshiya Maeda*,
Nachai Limsettho†, Keisuke Fujino†, Hideaki Hata†, Akinori Ihara† and Kenichi Matsumoto†
*Graduate School of Systems Engineering, Wakayama University, Wakayama, Japan
Email: {masao, s141015, s151049, s151054, s161053}@sys.wakayama-u.ac.jp
†Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan
Email: {nachai.limsettho.nz2, fujino.keisuke.fz9, hata, akinori-i, matumoto}@is.naist.jp

***Abstract*—The importance of supporting test and maintenance activities in software development has been increasing, since recent software systems have become large and complex. Although in the field of Mining Software Repositories (MSR) there are many promising approaches to predicting, localizing, and triaging bugs, most of them do not consider impacts of each bug on users and developers but rather treat all bugs with equal weighting, excepting a few studies on high impact bugs including security, performance, blocking, and so forth. To make MSR techniques more actionable and effective in practice, we need deeper understandings of high impact bugs. In this paper we introduced our dataset of high impact bugs which was created by manually reviewing four thousand issue reports in four open source projects (Ambari, Camel, Derby and Wicket).**

I. Introduction

The importance of supporting test and maintenance activities in software development has been increasing, since recent software systems have become large and complex. In the field of mining software repositories (MSR), several promising approaches have been proposed to help developers, for instance, to triage bugs [1], detect duplicate bugs [2], localize defects [3] and so forth.

Previous studies, however, have treated each bug equally without considering its impacts on bug management process and software products. For instance, a bug triaging method can predict *who should fix a bug* [1] but it can neither predict *who should fix the most important bug* nor *which bug would result in a deterioration in user satisfaction*. To make MSR techniques more actionable and effective in practice, we need much deeper understandings of high impact bugs.

Our pilot study [4] classified bugs reported to four open source projects into six types of high impact bugs [5]–[11], based on literature reviews. In the study one hundred bug reports were manually inspected for each project. The results of the investigation showed distributions of high impact bugs in the four projects and overlapped relationships among high impact bugs. Since the number of high impact bugs included in the dataset was very small, further investigations using a much larger sample size of data was required to improve the reliability of the conclusions. This paper introduces a large dataset of high impact bugs which includes manually-labeled four thousand issue reports.

II. Definitions of high impact bugs

Since a bug can highly impact on a variety of activities in the bug management process, products and end-users, several recent studies [5]–[11] have started with exploring high impact bugs such as performance, security, blocking bugs and so forth. In this paper, we roughly classify high impact bugs defined by the existing studies into two types: process and product. For instance, a blocking bug sometimes requires developers to reschedule bug-fix tasks due to its dependency on other bugs which must be fix prior to fixing the blocking bug. A performance bug can directly decrease users' satisfaction with products. In other words, a bug impacted on a bug management process is a matter to developers and a bug impacted on products is a matter to users. In what follows, we marshal studies on high impact bugs in terms of process and products.

A. Process

A bug can impact on a bug management process in a project. When an unexpected bug is found in an unexpected component, developers in the projects would need to reschedule task assignments in order to give first priority to fix the newly-found bug.

1) Surprise bugs: A surprise bug [5] 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, [5] 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 [6] 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).*" [6] 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'

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.78

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 [7]. 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 [7] (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).

B. Products

Bugs impacted on software products include security bugs [8], performance bugs [9], and breakage bugs [5]. They directly affect user experience and satisfaction with software products.

1) Security bugs: A security bug [8] 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.

2) Performance bugs: A performance bug [9] 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 [12]. [9] showed that a performance bug needs more time to be fixed than a non-performance bug. So performance bugs can affect users for a long time.

3) 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.

III. Data Collection

This section describes how we collect our dataset from issue reports in four open source projects. Issue report data has been collected from the Apache Ambari[1], Camel[2], Derby[3], and Wicket[4] projects where JIRA[5] is used for managing reported issues. These projects were selected because they met the following criteria for our project selection.

1) **Target projects have a large number of (at least several thousand) reported issues.** Our previous study [4] created a dataset consisting of four hundred issue reports in four open source projects (i.e., one hundred issue reports per project). A result of the study showed that the dataset only had a small number of high impact bugs which implied that it is difficult to use the dataset for prediction model building and/or machine learning in the future work. This is the reason why we would like to re-create the dataset at the moment.
2) **Target projects use JIRA as an issue tracking system.** From a project using JIRA, the information of AFFECTED and FIXED versions of an issue can be extracted. The version information is required to determine *Surprise* and *Dormant* bugs. Using issues reported to JIRA makes our investigation easier because only the rest of the other four types of bugs should be reviewed manually. We decided to select target projects supported by the Apache Software Foundation (ASF), because many of projects in the ASF use JIRA for tracking issues.
3) **Target projects are different from each other in application domains.** The distribution of high impact bugs was expected to be very different in application domains. At least twelve projects in the ASF met the first criterion above. After examining what they are developing, we decided to select the Ambari, Camel, Derby, and Wicket projects.

[1]The Apache Ambari project: http://ambari.apache.org/
[2]The Apache Camel project: http://camel.apache.org/
[3]The Apache Derby project: http://db.apache.org/derby/
[4]The Apache Wicket project: https://wicket.apache.org/
[5]JIRA: https://www.atlassian.com/software/jira/

TABLE I
DATA SOURCES AND OUR DATASET

	All the issues in Nov. 20 2014	Our Dataset	
		BUG	IMPROVEMENT
Ambari	8,389	871	129
Camel	8,063	580	420
Derby	6,772	734	26
Wicket	5,769	663	337

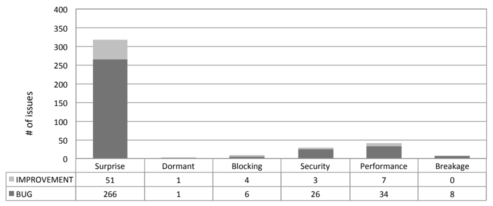
In JIRA, an issues report has a variety of types such as BUG, IMPROVEMENT, DOCUMENTATION, TASK, and so forth. Since we would like to focus on high impact bugs, we randomly selected one thousand issues with BUG or IMPROVEMENT from all the issues in each project. In general, IMPROVEMENT issues are not recognized as BUG, but we included them in the dataset because in our previous study they sometimes seemed to be equally treated as BUG. In fact, some of them could be classified into high impact bugs. In this paper we refer to a bug reported as an issue without distinguishing whether BUG or IMPROVEMENT.

Table I shows the number of all the issues in November 20 2014 and the number of BUG and IMPROVEMENT issues included in our dataset (one thousand issues per project).

IV. Manual Classification

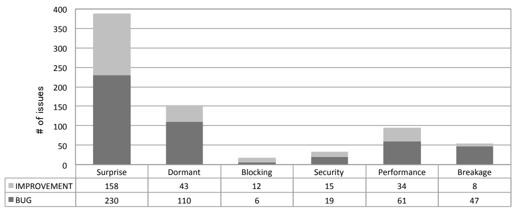
Since *Surprise* and *Dormant* bugs are easily detected based on the definition [5], [6] and can be automatically labeled on issues by using our script, the rest kinds of high impact bugs (i.e., *Blocking*, *Security*, *Performance*, and *Breakage* bugs) must be labeled manually. The manual classification was conducted as the following steps:

1) For a project, a graduate student of the authors reviewed a thousand issue reports and labeled one or more bug



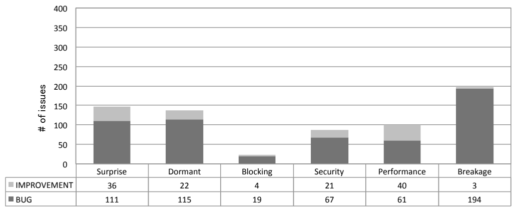
	Surprise	Dormant	Blocking	Security	Performance	Breakage
▨ IMPROVEMENT	51	1	4	3	7	0
▨ BUG	266	1	6	26	34	8

(a) Ambari



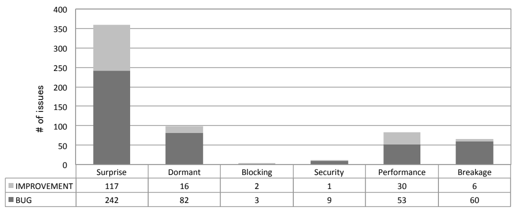
	Surprise	Dormant	Blocking	Security	Performance	Breakage
▨ IMPROVEMENT	158	43	12	15	34	8
▨ BUG	230	110	6	19	61	47

(b) Camel



	Surprise	Dormant	Blocking	Security	Performance	Breakage
▨ IMPROVEMENT	36	22	4	21	40	3
▨ BUG	111	115	19	67	61	194

(c) Derby



	Surprise	Dormant	Blocking	Security	Performance	Breakage
▨ IMPROVEMENT	117	16	2	1	30	6
▨ BUG	242	82	3	9	53	60

(d) Wicket

Fig. 1. Distributions of high impact bugs in four open source projects

types on each issue (i.e., multiple labelling is allowed.). Four graduate students participated in this session which took ten days to two weeks.

2) Four faculty members of the authors independently did the same thing as the students.
3) A student and faculty member who reviewed the same issue reports discussed differences of labeling between them until reaching a common understanding and labeling the same types on a single issue.

Although the review results were almost the same among students and faculty members, some of bugs were judged differently. In particular, the reviewers had to judge *performance*, *security* and *breakage* bugs subjectively to some extent since there are no established definitions to detect them while *blocking* can be semi-automatically identified by the definition in previous study [7].

TABLE II
INFORMATION INCLUDED IN OUR DATASET

NAME	Info.
issue_id	Issue ID
type	Type of an issue (BUG or IMPROVEMENT)
status	Status of an issue (Resolved or Closed)
resolution	Resolution type of an issue (FIXED only)
component	Target component/s
priority	Priority of an issue
reporter	Repoter's name
created	Time and Day of an issue reported
assigned	Time and Day of an issue assigned
assignee	Assignee's Name
resolved	Time and Day of an issue resolved
time_resolved	Time to resolve an issue (created to resolved)
time_fixed	Time to fix an issue (assigned to resolved)
summary	Summary of an issue
description	Descriptions of an issue
affected_version	Versions affected by an issue
fixed_version	Versions of a fixed issue
votes	Number of votes
watches	Number of watchers
description_words	Number of words used in descriptions
assignee_count	Number of assignees
comment_count	Number of comments for an issue
commenter_count	Number of developers who comment on an issue
commit_count	Number of commits to resolve an issue
file_count	Number of committed files to resolve an issue
files	Committed file names and paths

Figure 1 shows distributions of high impact bugs in the four open source projects. The number of *surprise* bugs are much larger than the other types of bugs except for Derby. In Derby, *breakage* bugs seems to be dominant issues. The reviewers for Derby pointed out that results of regression tests were actively reported to Derby's JIRA. The number of the other type of bugs are also moderately large except for *blocking* bugs. This might be because the Derby project has been developing a JAVA RDBMS which requires high reliability and performance and it also has a long history (i.e., large-scale and complex software).

Except for *surprise* bugs, *performance* bugs are relatively more than the other type of bugs throughout all the projects. Software performance seems to be considered very important across the projects because it directly affects users' operations and satisfaction with software products. Note that *performance* bugs were especially difficult to be captured because there is no explicit criteria to judge whether an issue represents a *performance* bug or not. The interpretations of *performance* bugs were often different among the reviewers although they had a session to reach a common understanding. In fact, the meaning of *performance* differed from application domains. We might need to further elaborate the dataset in the future.

Many *Dormant* bugs were also observed except for Ambari. In Ambari, *blocking* and *breakage* bugs are relatively less than *security* and *performance* bugs. This is probably because Ambari is the newest project among the target projects.

V. Other Information in the Dataset

Not only labeled information of high impact bugs but also other many information is included in our dataset. Since col-

TABLE III
MEDIAN DAYS TO FIX A BUG

	Sur	Dor	Blo	Sec	Per	Bre
Ambari	0.1	19.7	10.4	0.1	0.2	1.1
Camel	0.6	0.5	1.6	2.7	1.0	0.6
Derby	24.6	27.9	11.9	27.1	16.8	15.0
Wicket	2.8	1.2	6.9	0.8	3.7	3.8

TABLE IV
DISTRIBUTIONS OF PRIORITY

		Sur	Dor	Blo	Sec	Per	Bre
Ambari	Critical	29	0	0	3	8	0
	Blocker	5	0	2	1	2	2
	Major	276	2	8	25	30	5
	Minor	7	0	0	0	1	0
	Trivial	0	0	0	0	0	1
Camel	Critical	10	8	2	1	7	1
	Blocker	2	0	0	1	0	0
	Major	227	92	9	16	65	39
	Minor	137	50	6	13	21	15
	Trivial	11	3	1	2	1	0
Derby	Critical	3	4	3	0	3	6
	Blocker	1	3	1	0	1	4
	Major	76	85	14	61	57	140
	Minor	59	36	4	27	33	44
	Trivial	8	9	1	0	7	3
Wicket	Critical	7	1	0	1	4	5
	Blocker	0	0	1	0	1	0
	Major	227	72	2	8	62	45
	Minor	103	23	1	1	15	14
	Trivial	22	2	1	0	1	2

lected issue reports include a wealth of information regarding issues, we extracted as much information as possible. Table II shows a list of the information we extracted from issue reports. Our dataset is available from **http://goo.gl/r53j7w** as Microsoft Excel format.

Using these information together with the labels allows us to further investigate consequences of high impact bugs on a bug management process. For instance, Table III shows median days to fix a bug which vary from not only the projects but also the types of high impact bugs. Table IV shows distributions of priority of issues. "Major" is dominant priority throughout the projects. However there are many "Minor" issues except for Ambari, though we labeled issues based on the definitions of "high" impact bugs in the previous study. By only focusing on higher priority issues (i.e., Major, Blocker, and Critical), We might be able to achieve a new insight on "really high" impact bugs.

The dataset also can be used to analyze overlapping high impact bugs since multiple labelling was allowed. As we did in our pilot study [4], analyzing overlapping bugs (e.g., *security* and *performance* bugs) might bring an important clue to understand what is really high impact bugs for developers and/or users. The dataset also might be able to be used to improve or enhance existing MSR techniques. Although it is well-known that the information of priority and severity tagged on issues is often not reliable (i.e., the majority of priority and severity tags are middle-level.), many of existing techniques rely on such problematic information and never considered the practical importance of every single bug.

VI. CONCLUSION

This paper introduced our dataset of high impact bugs which was created by manually reviewing four thousand issue reports from four open source projects: Ambari, Camel, Derby and Wicket. In the future, we would like to further analyze high impact bugs using the dataset, improve the existing MSR techniques, try to build a new prediction model, for instance, to suggest who should fix the highest impact bug, and so on.

ACKNOWLEDGMENT

This work is conducted as part of Grant-in-Aid for Scientific Research: (C) 24500041, Grant-in-Aid for Challenging Exploratory: 26540029, Grant-in-Aid for Young Scientists: (B) 25730045, and the Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers by Japan Society for the Promotion of Science.

REFERENCES

[1] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, pp. 10:1–10:35, Aug. 2011.

[2] 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 (ICSE '10) - Volume 1*, 2010, pp. 45–54.

[3] P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: A literature review," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 5, pp. 1–8, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2659118.2659125

[4] Y. Kashiwa, H. Yoshiyuki, Y. Kukita, and M. Ohira, "A pilot study of diversity in high impact bugs," in *Proceedings of 30th International Conference on Software Maintenance and Evolution (ICSME2014)*, 0 2014, pp. 536–540.

[5] 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 (ESEC/FSE '11)*, 2011, pp. 300–310.

[6] 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 (MSR '14)*, 2014, pp. 82–91.

[7] 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 (MSR '14)*, 2014, pp. 72–81.

[8] 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 (MSR '10)*, 2010, pp. 11–20.

[9] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, 2013, pp. 237–246.

[10] 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 (MSR '11)*, 2011, pp. 93–102.

[11] A. Misirli, E. Shihab, and Y. Kamei, "Studying high impact fix-inducing changes," *Empirical Software Engineering*, pp. 1–37, February 2015, published online.

[12] I. Molyneaux, *The Art of Application Performance Testing : Help for Programmers and Quality Assurance*. O'Reilly Medea, 2009.