A Case Study on the Misclassification of Software Performance Issues in an Issue Tracking System

Masao Ohira
Faculty of Systems Engineering
Wakayama University, JAPAN
Email: masao@sys.wakayama-u.ac.jp

Hayato Yoshiyuki
Graduate School of Systems Engineering
Wakayama University, JAPAN
Email: s151054@sys.wakayama-u.ac.jp

Yosuke Yamatani
Graduate School of Systems Engineering
Wakayama University, JAPAN
Email: s151049@sys.wakayama-u.ac.jp

***Abstract*—In this study we focus on the misclassification of issue reports regarding software performance in OSS development. IEEE Std 610 [1] defines performance as "The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage." The definition implies that there is no rigorous criteria nor quantitative measure to detect a software performance problem, but rather it relies on a subjective judgement. OSS users sometimes mistakenly report a performance bug as a request for improving software performance and sometimes mistakenly report an improvement request for fixing a performance bug. The misclassification of report types can impedes the efficient bug-fix process in OSS development, since OSS developers preferentially spend time fixing bugs which might be improvement requests (i.e., not bugs) in reality. In this paper we strengthen our previous study for the Apache Wicket project by manually inspecting 1,000 bugs and 1,000 improvement requests respectively to more precisely understand the impacts of the misclassification on the bug-fix process in OSS development.**

1. INTRODUCTION

Open source software (OSS) is used not only for personal use but also for commercial products nowadays. As OSS users including developers of commercial products increases, a number of bugs are filed to an issue tracking system (ITS) every day in a large-scale project [2]. In general, any users can report a bug to ITS as an issue report when they encounter a problem in using OSS. The quality of a filed report often varies among users, since their understandings of computer systems including OSS differ among them. A good report helps developers fix a bug quickly and correctly, but a poor report arises many problems [3]. For instance, reproducing a bug is difficult [4] when a user does not report the information which is needed by OSS developers to resolve the bug. It is a time-consuming task because the developers must ask many questions to the user to reproduce the bug. Duplicate reports [5] also consume developers' time and effort. They are filed again and again because many users do not search similar reports which have been already resolved in the past before reporting a bug they found. In this way, the process of fixing bugs is sometimes far from efficient, despite a large number of bugs must be fixed before the next release.

In this study, we focus on the misclassification of issue reports regarding software performance. In IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610) [1], **performance** is defined as *"The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage."* The definition implies that there is no rigorous criteria nor quantitative measure to detect a software performance problem, but rather it relies on a subjective judgement. It is not so easy for every OSS user to judge if an encountered problem is really due to a performance bug. The misjudgments would lead to misclassify issue reports into improvement requests. OSS users sometimes report a performance bug as a request for improving software performance and sometimes report a improvement request as a bug. The misclassification of issue reports does not only degrade the reliability of defect prediction models [6], but also impedes the efficient bug-fix process in OSS development, since OSS developers preferentially spend time fixing bugs which might be improvement requests (i.e., not bugs) in reality.

In this paper we strengthen our previous study [7] where 1,000 bugs and 377 improvement requests in the Apache Wicket project were manually inspected to find the misclassification of issue reports. In this paper we use 1,000 bugs and 1,000 improvement requests respectively to answer the following research questions.

RQ1: *What is the impact of the misclassification on the bug-fix process?*
RQ2: *Why is the misclassification occurred?*

In what follows, Section 2 introduces types and purposes of issue reports managed by ITS. Section 3 describes the methodology of our study. Section 4 shows the result of our case study to answer the research questions. Section 5 discusses how to avoid the misclassification of issue reports. Section 6 summarize the paper and describes our future work.

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE
ICIS 2016, June 26-29, 2016, Okayama, Japan

TABLE 1. ISSUE TYPE IN AN ISSUE TRACKING SYSTEM

Type	Purpose
Bug	To manage to fix reported bugs.
Improvement	To manage requests for improving software.
New Feature	To manage requests or proposals for new features.
Test	To manage proposed test cases.
Task	To manage tasks related to a project.

2. TYPES OF ISSUE REPORTS

An issue tracking system such as Bugzilla[1] and Jira[2] is used not only for managing reported bugs, but also for managing various types of information (i.e., issues) related to an OSS project. Table 1 shows types of issues which are reported by users and managed by ITS. **Bug** is used to manage and track the information from bug reporting to bug resolution. **Improvement** is used to manage requests for enhancements and extensions of existing features. It also includes improvement requests for concerns which are not exposed at the time but would cause problems in the future. **New Feature** manages requests and proposals for new features and **Test** is for proposals of test cases. **Task** is used to manage general tasks in a project. It is also used to manage tasks that should be preprocessed before dealing with reported bugs and improvement requests.

The type of an issue is labeled by a reporter when s/he files an issue with ITS. It is basically used until the issue is resolved (closed), though it is sometimes changed along the way. As is the case with the problems of duplicate and not-reproducible bugs reported by users who are not familiar with software systems, a performance bug could cause a problem since it might be mistakenly filed as **Improvement** by casual users who do not have enough knowledge to judge if it should be filed as **Bug** or **Improvement**. An issue which is filed as **Bug** but should be **Improvement** might consume developers' time and effort because developers would try to fix it preferentially. In contrast to the misclassification of **Improvement**, an issue which is filed as **Improvement** but should be **Bug** would be also problematic since an improvement request would not necessarily need to be addressed quickly, that is to say, a performance bug might be led for later or be missed. These potential problems motivate us to study the misclassification of **Bug** and **Improvement** in reporting performance problems.

3. STUDY METHODOLOGY

3.1. Selecting a Project: Apache Wicket

In this study we investigate the misclassification of performance issue reports in the Apache Wicket project. The project has been developing a Java framework for Web applications since 2005. The Apache Wicket project is selected for our study mainly because of the following reasons: (1) Wicket is a part of a development environment for web application developers and therefore developers would attach importance to performance. (2) Users would attach importance to performance of Web applications created by using Wicket. (3) Regarding a performance problem encountered in using a Wicket-based Web application, it would be difficult to identify if the root cause is due to Wicket or due to a computational environment where the application is used. Besides the above reasons, Wicket would be an appropriate subject to be investigated since Wicket is an Apache top-level project, has been being developed vigorously and has sufficient data (i.e., a number of issues resolved in the past). Table 2 shows the number of **Bug** and **Improvement** issues filed to ITS in the Wicket project as of January 2016.

1. Bugzilla: https://www.bugzilla.org
2. Jira: https://www.atlassian.com/software/jira

TABLE 2. NUMBER OF BUGS AND IMPROVEMENTS IN WICKET (AS OF JANUARY 2016)

Type	Issues
Bug	3,728
Improvement	1,703

TABLE 3. DIFFERENCES OF DATASET BETWEEN PREVIOUS STUDIES [8] [7] AND THIS STUDY

Issues	[8]	[7]	This paper
Number of Bugs	663	1,000	1,000
Number of Improvements	337	337	1,000

TABLE 5. PERFORMANCE ISSUES INCLUDED IN OUR DATASET

Type	Issues	Perf. Issues
Bug	1,000	61
Improvement	1,000	59

3.2. Preparing Dataset

Issue data used in this paper is originally collected by Ohira et al. [8] who randomly sampled 1,000 issues from the Apache Ambari, Camel, Derby and Wicket projects respectively. The original 1,000 issues for each project consist of both Bug and Improvement issues in order to investigate problems which highly impact on OSS development and user satisfaction[3]. For each issue, types of problems (e.g., performance and security) are labeled manually.

In [7], issue data was added to [8] only for the Wicket project as the dataset consisted of 1,000 Bugs. In this paper, we further add issue data to [7] as the dataset also has 1,000 Improvements to be analyzed, in order to increase the generality of the empirical findings in [7]. We believe that it would be worth investigating much more misclassified Improvements, because misclassifying Bug issues as Improvement issues would lead to consume developers' time and effort. Table 3 shows the differences of the datasets

3. High Impact Bug Dataset: http://oss.sys.wakayama-u.ac.jp/?p=1009

TABLE 4. TYPES OF SOFTWARE MAINTENANCE ACTIVITIES [9] AND CORRESPONDING ISSUE TYPES IN THIS STUDY

Type of Maintenance	Descriptions of Activities	Issue Type
Corrective	Modifications of software products to correct problems or failures found after delivery	Bug
Adaptive	Modifications of software products to adapt to changes of computational environments (e.g., OS upgrade) as the products can be used for a long time	
Perfective	Modifications of software products to improve usability, performance, maintenability and so forth which are not problems in the current situation	Improvement
Preventive	Modifications of software products to correct potential problems after delivery, which do not become apparent in the current situation	

between previous studies [8] [7] and this study. As is the case with the previous studies, added issue data is randomly selected and met with the following criteria.

- An issue is resolved (i.e. not work in progress).
  - **Status** of the issue is **Resolved** or **Closed**.
  - **Resolution** of the issue is **Fixed**.
- An issue has been actually treated as **Bug** or **Improvement** (i.e., It is not **Closed** without anything).
  - An issue has one or more versions in **Affects Version/s**.
  - An issue can be linked to a commit log which indicates that code modifications are actually needed for fix the issue.

3.3. Judging Misclassifications

Although some of issue reports in our dataset are labeled as **Performance** which means that the issues report performance problems, we still need to judge manually if they should be treated as Bug or Improvement. However, as we described earlier, there are no rigorous criteria to do so. In order to mitigate this problem, we use the types of software maintenance activities described in ISO/IEC 14764:2006 [9] to judge if an issue report including a performance problem should be classified into Bug or Improvement. Table 4 shows the types of software maintenance activities and corresponding issue types (i.e., Bug or Improvement) in this study. According to [9], software maintenance is classified into four types of activities with different purposes and goals: Adaptive, Corrective, Perfective and Preventive.

Of the four types of software maintenance activities, the corrective and adaptive maintenance aim at resolving problems happened by programming errors, changes of computational environments, and so forth. Therefore, we can consider that the two activities cover issues which accompany concrete symptoms or signs of problems. By contrast, the perfective and preventive maintenance aim at resolving potential problems which are not observed in the current situation but would be appeared in the future. That is to say, they cover issues which should be hopefully fixed in the near future. In this study, the corrective and adaptive maintenance are regarded as maintenance activities for fixing Bugs and the perfective and preventive maintenance are regarded as maintenance activities for coping with Improvements.

TABLE 6. MAINTENANCE ACTIVITIES FOR FIXING BUG ISSUES (**bold** TYPES ARE MISCLASSIFIED BUG ISSUES)

Type of Maintenance	Issues
Corrective	51
Adaptive	4
Perfective	**5**
Preventive	**1**

Table 5 shows the number of issues with performance problems in our dataset. 61 of 1,000 Bug issues and 59 of 1,000 Improvement issues have problems with software performance. In other words, about 6% of issues reported to the Wicket project relates to software performance, regardless of whether they are Bug or Improvement. Based on the above definition (Table 3), we manually review the 120 issues (61 Bugs and 59 Improvements) to identify the misclassification of issues relating to software performance.

4. Results

4.1. Misclassification of Bug Issues

Table 6 shows a result of manual reviews for Bug issues which are labeled as performance problems. Of the 61 Bug issues, 55 (90.1%) issues were correctly classified. The corrective maintenance was required to fix 51 (83.6%) issues which reported obvious problems such as memory leak. The adaptive maintenance was required only for 4 (6.5%) issues which reported performance degradation after updating Wicket, awkward behaviors in particular browsers and so forth.

In contrast with the correctly classified Bug issues, 6 (9.8%) Bug issues required the perfective or preventive maintenance to be fixed. In other words, the 6 issues should be treated as Improvement issues, but they were misclassified as Bug issues. The perfective maintenance was required for five Bug issues and the preventive maintenance required for one Bug issue. Table 7 shows overviews of misclassified Bug issues. In what follows, some examples of the misclassified Bug issues are described.

TABLE 7. OVERVIEWS OF MISCLASSIFIED BUG ISSUES

Issue ID	Type of Maintenance	Overview
WICKET-1684	Perfective	User request for changing a return value
WICKET-2025	Perfective	Unnecessary method call
WICKET-2386	Perfective	Change of JavaDoc
WICKET-3207	Perfective	A method is called twice
WICKET-5527	Perfective	Slowness due to inefficient use of a class
WICKET-3669	Preventive	Same JavaScript code is run twice

TABLE 8. OVERVIEWS OF MISCLASSIFIED IMPROVEMENT ISSUES

Issue ID	Type of Maintenance	Overview
WICKET-315	Corrective	Very slow parsing for web.xml
WICKET-1175	Corrective	Integer overflow
WICKET-1502	Corrective	Slowness in opening complex pages
WICKET-1790	Corrective	Resource waste and slow performance
WICKET-1910	Corrective	Performance problem due to no cash mechanism
WICKET-4270	Corrective	Lightweight operation is needed
WICKET-4285	Corrective	Bad exception handling wasting resources
WICKET-4554	Corrective	Unnecessary directory creation every time test fails
WICKET-5933	Corrective	Serialized same data again and again

4.1.1. Examples of misclassified Bugs (Perfective). WICKET-2025[4] reports an unnecessary method call to be fixed. Since the problem is only observed in the development mode of Wicket and does not appear in the deployment mode (i.e., few users affect the problem), WICKET-2025 is regarded as an Improvement issue which should be resolved in the perfective maintenance.

WICKET-3207[5] reports that the same method is unnecessarily called twice and it might lead to problems in some situations. Since it seems that concrete problems were not observed when the issue was reported, but calling the same method twice consumes computational resources, WICKET-3207 is regarded as an Improvement issue which should be resolved in the perfective maintenance.

WICKET-5527[6] reports an inefficient cash use which leads to the performance degradation. Since the problem is reported by a developer and users do not seem to notice the problem, in this study WICKET-5527 is a performance bug but is regarded as an Improvement issue.

4.1.2. An example of a misclassified Bug (Preventive). WICKET-3669[7] reports that a JavaScript code is executed twice in specific web browsers. It is considered as a Bug issue which should be fixed to adapt to changes of users' computational environments (i.e., a bug to be fixed in the adaptive maintenance), but it is also regarded as an Improvement issue because it does not seem to strongly impact on users. In fact, WICKET-3669 took 229 days to be fixed.

4. https://issues.apache.org/jira/browse/WICKET-2025
5. https://issues.apache.org/jira/browse/WICKET-3207
6. https://issues.apache.org/jira/browse/WICKET-5527
7. https://issues.apache.org/jira/browse/WICKET-3669

TABLE 9. MAINTENANCE ACTIVITIES FOR FIXING IMPROVEMENT ISSUES (**bold** TYPES ARE MISCLASSIFIED IMPROVEMENT ISSUES)

Type of Maintenance	Issues
Corrective	**9**
Adaptive	**0**
Perfective	46
Preventive	4

4.2. Misclassification of Improvement issues

Table 9 shows a result of manual reviews for 59 Improvement issues which are labeled as performance problems. Of the 59 Improvement issues, 50 (84.7%) issues were correctly classified: the perfective maintenance was required for 46 (78.0%) issues and the preventive maintenance was required only for only 4 (6.7%) issues. In contrast with the correctly classified Improvement issues, 9 (15.3%) issues required the corrective maintenance to be fixed. In other words, the 9 issues should be treated as Bug issues, but they were misclassified as Improvement issues. There were no issue which required the adaptive maintenance. Table 8 shows overviews of misclassified Improvement issues. In what follows, some examples of the misclassified Improvement issues are described.

4.2.1. Examples of misclassified Improvements (Corrective). WICKET-315[8] reports that parsing web.xml with the DOM API is very slow. A developer commented that it can be around 200 times faster by using other parsers. Since the problem affects a wide range of users, it should be treated as a Bug which is resolved in the corrective maintenance.

8. https://issues.apache.org/jira/browse/WICKET-315

TABLE 10. MEDIAN DAYS TO ASSIGN AND FIX AN ISSUE

	Type	Issues	Time to assign (days)	Time to fix (days)
Bug	Correctly classified	55	0.400	0.318
	Misclassified	6	7.477	6.782
Improvement	Correctly classified	50	3.589	0.432
	Misclassified	9	8.275	25.527

WICKET-1790[9] reports a strong need for optimizing javascript which leads to slow down page rendering significantly. Since this problem also affects many users, it should be treated as a Bug which is resolved in the corrective maintenance.

The other Improvement issues also reports problems of wasting resources and slowness which have an impact on the user experience. Since they are explicitly observed in user's environments, they should be also treated as Bugs which should be resolved in the corrective maintenance.

5. DISCUSSIONS

As described in the previous section, we inspected 61 Bug issues and 59 Improvement issues which were reported to resolve performance problems. We then found that 6 of 61 (9.8%) Bug issues and 9 of 59 (15.3%) Improvement issues were misclassified. We believe that the ratio of misclassified issues is not so low to be ignored, since developers would preferentially try to spend their time to fix bugs which might be just "Improvement" requests while developers might neglect Improvement issues which are misclassified "Bugs" to be fixed quickly. In this section, we investigate and discuss actual impacts of the misclassification on the software maintenance process.

5.1. Impacts of the misclassification

In our manual reviews for Bug and Improvement issue reports including performance problems, we found that some misclassified issues needed longer time to be resolved than correctly classified issues. In order to more precisely understand the impact of the misclassification on the bug-fix process, we analyzed time to assign and resolve an issue: the time to assign a task to a developer from when an issue is reported and the time to complete the task (i.e., the time to resolve the issue) from the task is assigned. Table 10[10] shows the result of the additional analysis.

Of the 61 Bug issues, the correctly classified 55 Bug issues were assigned and fixed within a day. By contrast, the 6 misclassified Bug issues (i.e., the truth is Improvement issues) spent over 7 days to be assigned and spent over 6 days to be resolved. As we showed in the overviews of the 6 issues (Table 7), these issues included neither urgent nor concrete problems affecting users widely. Developers in the Wicket project seemed to be aware of the impacts of these issues on the users (i.e., they are not so important problems to be fixed quickly). In that context, the developers properly dealt with the 6 issues, although they were misclassified as Bugs.

Of the 59 Improvement issues, the correctly classified Improvement issues required about 3.6 days to be assigned, but they were fixed within a day after the assignments. Although compared with the correctly classified 55 Bug issues, these issues needed more time to be assigned, Improvement issues in our study should be addressed in the perfective or preventive maintenance where a problem does not need to be resolved urgently. By contrast, the 9 misclassified Improvement issues (i.e., the truth is Bug issues) spent over 8 days to be assigned and surprisingly spent over 25 days to be resolved. These Improvement issues should be quickly fixed as Bugs, since the reported problems can be observed and could affect many users. From the result of the additional analysis, we can consider that the misclassification of Improvement issues might lead to prolong the issue resolution.

RQ1: *What is the impact of the misclassification?*
Misclassified issues could increase the time to assign and fix the issues. Especially, bugs misclassified as Improvement issues would be left unresolved for a long time.

5.2. Why Misclassified?

To understand why the misclassified issues were created, we analyzed what and how developers discussed issues in BTS. Table 11 shows the median number of comments and the median number of developers per performance issue. Comments for the misclassified Improvement issues are much larger than other types of issues. For instance, WICKET-1175[11] (Figure 1) reported an integer-overflow problem and involved 7 developers in the discussions consisting of 17 comments. In the discussions for WICKET-1175, one developer firstly mentioned that they could fix the problem, but developers decided not to fix it at the time since changes from Long to Integer would be needed for too many places of Wicket. A few years later, the developers discussed the same problem again and again. Finally, it took over 4 years to be resolved. From these results, we can consider that the misclassified Improvement issues were not mistakenly reported, but rather, they are not inevitable to

9. https://issues.apache.org/jira/browse/WICKET-1790

10. Please note that some issues were assigned and fixed simultaneously (i.e., the same time stamp is recorded on a single issue). It would be recorded in BTS since developers reported to BTS after they fixed issues. We did not include the simultaneously assigned and fixed issues to calculate the result.

11. https://issues.apache.org/jira/browse/WICKET-315

TABLE 11. MEDIAN NUMBER OF COMMENTS AND COMMENTATORS PER AN ISSUE

	Type	Issues	Num. of Comments	Num. of Commentators
Bug	Correctly classified	55	3	2
	Misclassified	6	2	1
Improvement	Correctly classified	50	2	2
	Misclassified	9	**8**	3

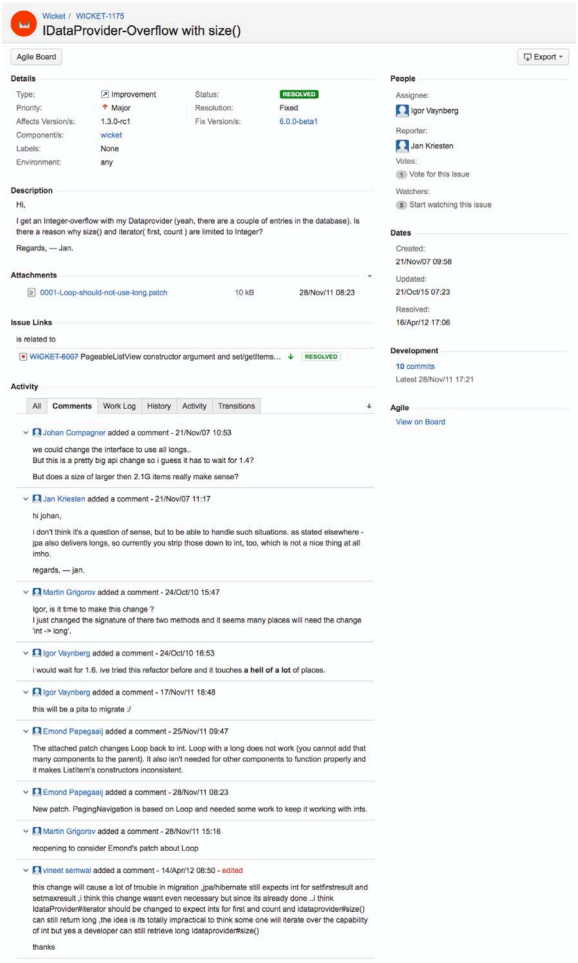

Figure 1. WICKET-1175

some extent because they need big code changes and so forth which cannot be handled easily and quickly. As shown in the previous subsection, the misclassified Improvement issues tend to prolong to be fixed. The data also supports the reason why the misclassified Improvement issues were reported and inevitable.

RQ2: *Why is the misclassification occurred?*
Misclassified Improvement issues would be filed because developers involuntarily have to leave them for long even if the developers are aware of them as Bugs to be fixed.

6. Conclusion and Future Work

In this paper we conducted a case study of misclassified performance issues in the Apache Wicket project. We manually inspected 1,000 Bug and 1,000 Improvement issues respectively to understand the impact of the misclassification on the bug-fix process and the reason why the misclassification occurred. However, we need to study more projects in the future to achieve general findings because we only focused on an OSS project.

Acknowledgments

We are very grateful to Yuta Matsuo for helping us prepare the dataset used in this study. This work is conducted as part of Grant-in-Aid for Scientific Research: (C) 24500041 by Japan Society for the Promotion of Science.

References

[1] IEEE Std 610, "Ieee standard glossary of software engineering terminology," IEEE.

[2] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*, 2009, pp. 111–120.

[3] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, Sept 2010.

[4] J. Bell, N. Sarda, and G. Kaiser, "Chronicler: Lightweight recording to reproduce field failures," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*, 2013, pp. 362–371.

[5] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*, 2012, pp. 70–79.

[6] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *Proceedings of the 37th International Conference on Software Engineering (ICSE '15)*, 2015, pp. 812–823.

[7] Y. Matsuo, H. Yoshiyuki, and M. Ohira, "The misclassification of performance bugs in oss projects: A case study of apache wicket," in *Proceedings of Software Symposium 2016 in Yonago (SS2016)*, 2016, p. (to appear) (in Japanese).

[8] 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 12th Working Conference on Mining Software Repositories (MSR 15)*, 5 2015, pp. 518–521.

[9] ISO/IEC14764:2006, "Software engineering–software life cycle processes–maintenance."