# A Case Study of Committers' Activities on the Bug Fixing Process in the Eclipse Project

Anakorn Jongyindee[†] Masao Ohira[‡] Akinori Ihara[‡] Ken-ichi Matsumoto[‡]
[†]Kasetsart University
50 Ngam Wong Wan Rd, Chatuchak
Bangkok, Thailand
b5105896@ku.ac.th

Nara Institute of Science and Technology
8916-5, Takayama, Ikoma
Nara, Japan
{masao, akinori-i, matumoto} @ is.naist.jp

## ABSTRACT

There are many roles to play in the bug fixing process in an open source software development. A developer called "Committer" who has permission to submit the patch into the software repository plays a major role in this process and holds a key to the success of the project. In this work, we have observed committers' activities from the Eclipse-Platform bug tracking system and version archives. Despite the importance of committer's activities, we suspected that sometimes committers can make mistake, which have a negative consequence to the bug fixing process. Therefore, our research focused on studying the consequences of committers' activities to this process. We collected committers' history data and evaluated each of them by comparing the more cautious to less cautious committers. From our results, we would like to create a clear understanding on committers' activities and their consequences on the bug fixing process in order to find a better way to improve the bug fixing process in OSS projects.

## Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Complexity measures, Performance measures*; D.2.9 [**Software Engineering**]: Management —*Productivity, Programming teams, Software quality assurance (SQA)*

## General Terms

Human Factors

## Keywords

open source software (OSS), committer, bug fixing process

## 1. INTRODUCTION

Open Source Software (OSS) has been attracting a great deal of attention from a variety of areas as an alternative way to use and develop software. Currently, OSS products has a large impact on not only the end-users, but the manufacturers of mobile devices as well, since they need to exploit OSS to produce their end products. For instance, Linux, the OSS that sets the strong roots for other operating systems, and Google's Android's operating system that, based on this OSS, has become the best-selling smart phone platform. As OSS become more common and popular among us, however, its projects are faced with a big challenge to their quality assurance activities. Due to their growing user base, especially large OSS projects such as the Mozilla and Eclipse projects receive considerable amount of bug reports from the users on a daily basis [13] (e.g., several hundred bug reports are posted to the Bugzilla database of the Mozilla project every day). Therefore, OSS projects require an effective way of dealing with the large number of bug reports.

In an OSS project, a bug is fixed through the bug fixing process [19] which is a process of fixing the bug from the time a bug was reported in the project until the patches for fixing the bug have been submitted into a software repository such as Bugzilla. Each bug report in this process is passed through one or more developers who play different roles before the process is closed.

In this study, we focused on developers who have the privilege to submit patches into the software repository, called *Committers*. This group of developers play major roles in the bug fixing process [7]. Their main task is to review (and sometimes edit) patches posted by other developers and submit them into the software repository. Some of them also perform other tasks including bug resolution and bug reports management. Using Concurrent Versions System (CVS) and Bug Tracking System (BTS), they resolve bugs, join discussions about bugs, verify fixed bugs by developers, close bug reports, and so forth. As just described, committers' activities are vital for sustaining and improving the quality of OSS products.

However, committers are not always perfect. They sometimes make mistakes. For instance, they can uncautiously verify a bug report already resolved by a developer and close the bug report, creating another bug report for the same bug again (i.e., reopen bug). In this paper we are interested in creating a better understanding of committers' activities and their consequences on the bug fixing process in order to find a better way to improve the bug fixing process in OSS projects.

Selecting the Eclipse-Platform's Version archives (CVS) and Bug Tracking System (BTS) as the information source of our case study, we ask the following research question.

**RQ: What are the consequences of the committer's activities to the bug fixing process?** In this question, we studied the committer's activities and their effects to the bug fixing process. Then we compared the consequence of more cautious committer's activities to the lesser one.
Through answering the research question, we provide contributions in this paper as follows.

- To create a better understanding on a committersÕ activities and their consequences on the bug fixing process in order to find a better way to improve the bug fixing process in OSS projects.

In what follows, we introduce our related works in Section 2 and extraction method in Section 3. Section 4 shows the results and process on how we answered our research question. Additional interesting results that we are able to identify during this work are discussed further in Section 5. Section 6 describes our limitations and future work, and we summarize our study in Section 7.

# 2. RELATED WORK AND MOTIVATION

Most existing studies are focused on how to reduce the time to fix bugs since it has been gradually increasing, especially in large OSS projects. There are currently three promising approaches to improving the bug fixing process. In what follows, we describe the existing approaches and our motivation of this study.

## 2.1 How to make a good bug report?

A good bug report contributes by reducing the time to fix bugs because it can help developers to quickly find, replicate, understand the bugs at hand. However, developers' information needs in the bug reports are often unsatisfied, since users do not know what type of information are required to fix a problem and so rarely articulate the problem on the software use as developers can fix it. For instance, users do not correctly report procedures to reproduce an error (e.g., sometimes they just say "This option does not work in my computer!"). Therefore, developers have to ask users to give more information again and again to identify and fix the error. If things go wrong, developers cannot confirm the error and then leave it unresolved reluctantly.

In order to improve cooperation on a bug report between developers and users, many studies [3, 4, 6, 10] have interviewed OSS developers and users to understand the information needed to fix the bug. For example, through interviews with over 150 developers and 300 reporters of the Apache, Eclipse and Mozilla projects, Bettenburg et al. [BettenburgFSE2008] have found that steps to reproduce and stack traces are most useful in bug reports.

## 2.2 Duplicate bug detection

Users often report the same problem that was reported by another user in the past or that has already been fixed by developers. Developers also sometimes try to resolve the same problem which had been resolved in other times. This can happen because there are a large number of bug reports in the bug tracking system. Both the users and developers cannot be aware of all the reported bugs though the searching function that was provided to find bugs reported in the past. In this manner, the same bugs are duplicated in BTS which result in wasting developers' time and efforts.

To avoid duplicate bugs in BTS, several studies [18, 15, 20] have tried to detect duplicate bug reports automatically. For example, Wang et al. [20] present an approach to detect duplicate bugs based on a natural language processing techniques.

## 2.3 Re-opening and reassigned bugs

Even if a bug fixing task is assigned to a developer, it may not be completed by the developer which is then reassigned (*tossed* [13]) to other developers. This often happens because a trigger assigned a bug fixing task to an inappropriate developer who does not have sufficient knowledge and skill to complete the task. In the Eclipse and Mozilla projects, 37% to 44% of bugs are reassigned to another developer [13]. Preventing the bug tossing (assigning a bug fixing task to appropriate developers) is very effective in reducing the time to fix bugs.

Several approaches [1, 14, 13, 9, 8] exist in this topic. For instance, Anvik et al. [1] proposed an approach to assign a bug to an appropriate developer based on past bug reports with natural language processing. Jeong et al. [13] also tried to establish a method for the bug assignment based on a social graph which reflects on social relationships among developers in the bug assignment. Other approaches involved in achieving better understanding on why reassignment occurs many times [9] and in creating a method to predict which bugs will be reopened or get fixed without being reopened [8].

## 2.4 Cautious and uncautious committers

In general, a dedicated developer is nominated or elected to become a committer in an OSS project [12]. The OSS project carefully selects a developer as a committer candidate since committers play the important roles as described earlier. It takes one or two years to be a committer. A developer who wish to be a committer has to keep showing devoted activities to the project for a considerable period of time. Due to this promotion process, many of developers leave the project within a year and the OSS projects are always faced with the difficulty in having to find more committers who can greatly contribute to the bug fixing process.

In order to find a way to increase committers in OSS projects, Fujita et al. [7] have examined activities of developers and committers in the PostgreSQL project and tried to identify promising developers who were making a significant contribution equivalent to committers and potentially should be nominated to be committers in the future. Although the study found interesting aspects of committer candidates, it still adhered the current practice of existing committers and their promotion process in OSS project. In fact, one of their conclusions was that long-term participation in a project was the most important aspect to become a committer.

Different from [7], in this paper we have studied the committers' activities and their consequences. Our basic assumption on committers is that committers are not always perfect and sometimes make mistakes because they are also human-beings. Some of them might uncautiously verified a fixed bug by developers, creating reopened bug reports in the future. They also might uncautiously reviewed and accepted a patch posted by developers to fix a bug and then committed it into the repository such as CVS that would bring reopen and another bug reports. In this study we are

interested in having a clear understanding of committers' activities and the consequence to the bug fixing process.

# 3. EXTRACTION METHOD

In this section, we describe how we extracted information of committers' activities from the Eclipse-Platform project for our case study. Firstly, we describe how records of committer's activities are preserved in OSS the development. Then we introduce our method of extracting the information to observe committer's support activities and main activities respectively.

## 3.1 Committers' Activities

When developers are involved in the bug fixing process, their action are recorded in many formats. In common with many other OSS projects, Eclipse-Platform had chosen Bugzilla as their BTS where records each committer's support activities such as patch reviews and status changes (e.g., sometimes developers check the resolution of a bug and mark the bug's status to "VERIFIED" or "CLOSED"[19]) are stored. Developers in the project can see the information of the database in the HTML form through a web browser.

The project also used CVS to keep track of their commit history which is recorded in the plain text format called "commit log". By combining the information from both CVS and BTS, we are able to observe when/why/how each developer had contributed in the bug fixing process.

Using both the commit log and the Bugzilla database as our data sets, we collected 85,387 bug report data on BTS and over 30,833 commit log data on CVS. As a result, we were able to captur activities of 2,584 different developers from October 2001 until January 2010.

In order to archive our results, first, we need to identify who the committers are, excluding them from thousand of regular developers in the projects. To our knowledge, there is no specific activity that can decide whether one developer is a committer or not. [7] suggests only a rough description of how they extract their committer list. Based from their work, they had defined a developer as a committer, who has a privilege to submit a patch to the software repository. By using this definition, we managed to extract our list of committer's names. When a committer makes a patch commitment, their action is captured and their name is recorded in commit log's author field. By using regular expressions to scan every CVS line, we are able to identified 74 privilege developer names to create our list of committers' names.

From the committer list, we need to collect each committer's behavior data. We describe the procedures in the following separate subsections. In the first subsection we described how we collect each committer's support jobs in BTS, that is, how we studied the footprint of their support activities left on Bugzilla[1]. The second section describes how we observe committer's main jobs in CVS, that is, how we collects their patch commitment footprint left on CVS data.

## 3.2 Observing Support Activities From BTS

[1]Only some committers use the same account name in CVS and BTS. In order to map between their two accounts, we used automated method to find an exact-name-match for the committer who use same account name in both records. For those who did not, we have no choice but to map each committer's name manually.

In the bug tracking system, each reported bug is identified by a number called bug-id, attached with other data such as bug priority, bug status history, developer's comment, and so on. Each bug has its own current status varying from NEW, ASSIGNED, VERIFIED or CLOSED. Some bug status have its own resolution to indicate what happened to the bug such as FIXED, INVALID, and DUPLICATED[19].

Bug status history are used in many researches as a very useful source of information. Researchers can test a hypothesis[7], create prediction models[16], or performs statistical analysis[11]. In this paper, when we describe the bug status that have changed from one to the others in the bug history, for better clarifications, we present the bug's history in the form of bug status patterns. We use "⇒" for separation between bug status. Time dimension flow from left to right of the patterns. "..." Symbols represent any or many bug status changed and we use "()" to show the resolution of the bug status if it is exist. These bug status pattern can start from as simple as OPENED ⇒ NEW ⇒ ASSIGNED ⇒ RESOLVED (FIXED) to the more complex pattern such as OPENED ⇒ NEW ⇒ ASSIGNED⇒ RESOLVED (INVALID) ⇒ REOPENED ⇒ ASSIGNED ⇒ RESOLVED (WORKSFORME). For the first pattern, we can observe that the bug has been assigned only once before its resolved. This type of the pattern usually leads to short or normal bug life cycle while the more complex one often leads to longer bug life cycle.

We are able to observe and collect each committer's support activities based on this bug status patterns. We could identified 52,013 of 85,387 bug reports that were involved by our committers with 4,941 of 30,833 difference bug status patterns.

## 3.3 Observing Main Activities From CVS

For the Eclipse-Platform's commit log (from CVS), we wish we could have observed the committer's main behaviors solely from the commit log as we did in the Bugzilla bug history. Unfortunately, from this commit log we can only have a narrow vision of committer's activities; It captures only revision numbers, date of commit and some information about source code changes. The description about each change is solely depend on committer's opinion. This description field has no centralized format and often recorded in an ill-organized pattern (some of them are empty sometimes).

Due to these inconsistencies, the CVS description is not adequate to judge each committer's behavior. In order to overcome this problem, we decided to adapt T. Zimmerman's[17] approach to our study. Their approach has suggested that, despite its inconsistencies, sometimes committer has mentioned bug-id in the CVS description. By using bug-id as a trails, we identified it as a *links* from the CVS repository to the bug database. From these *links* we can look further into the BTS database where we have wider behavior information to study. The technique on finding these *links* has been used in many works in this field (e.g., [2],[17] and [5] has described these links and illustrated it clearly.). By adapting the Zimmerman's approach to our study, we managed to identify 1,193 links from our commit log. Thus, we could collect each committer's main activities.

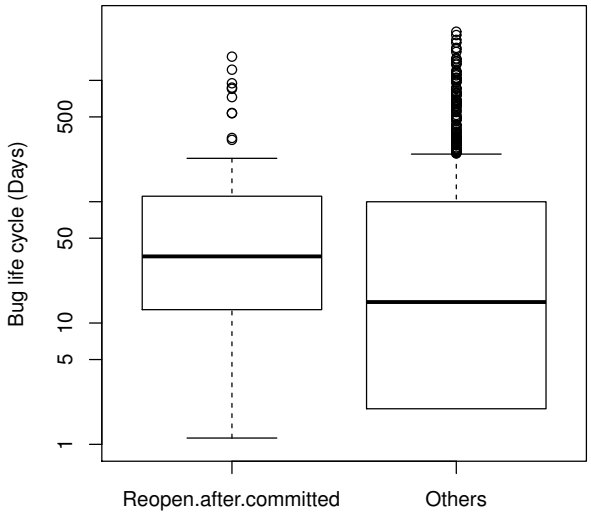
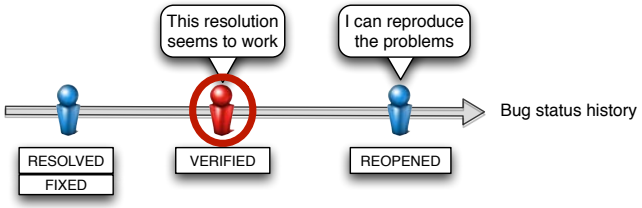
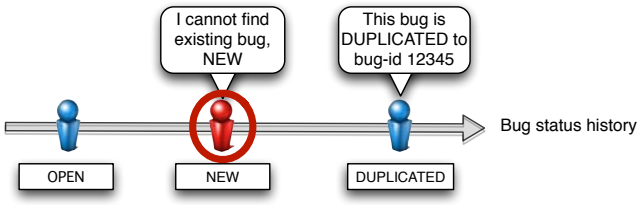



**Figure 1: Box plot comparing the life cycle of *Reopen-after-committed* bug and normal bug**




(a) *Reopen-after-verified/closed* pattern




(b) *Duplicate-after-new* pattern

**Figure 2: *Bad-status-patterns* observed in our study**

# 4. RESULTS

**RQ: What are the consequences of the committer's activities to the bug fixing process?**

By focusing on the consequences of committer's activities, we separated the results for this question into two parts: the consequences of a committer's main activities and the consequences of a committer's support activities.

## 4.1 Consequences of Main Activities

**APPROACH.**

As mentioned above, we suspect that when a committer un-cautiously committed the patch that fixed the bug, this bug might be reopened later to be resolved again. After compare the bug life cycle of these *Reopen-after-committed* bugs with the other bugs that did not reopened from 1,193 links found from CVS and BTS, we identified 140 bugs that had been reopened after committers committed the patches.

**FINDING.**

The result are shown in Fig 1, we can see that when a bug is reopened after patches were committed, the bug tends to have longer life cycles.

## 4.2 Consequences of Support Activities

**APPROACH.**

As explained earlier, we collected committer's support activities by observing the bug status history and rewrote these status history in the form of patterns. In order to study the consequence of these patterns to the bug fixing process, we had randomly chosen these patterns to inspect manually. By focusing only on patterns related to committer's jobs, we can identify two status patterns that potentially have negative effects on the bug fixing process.

The first pattern shown in Fig 2 (a) we call *Reopen-after-verified/closed* pattern represents that bugs have been REOPENED after they had been marked as VERIFIED or CLOSED (e.g., ... ⇒ VERIFIED (FIXED) ⇒ REOPENED or ... ⇒ CLOSED (FIXED) ⇒ REOPENED). We suspect that this pattern occurs when committers do not cautiously check a patch before they changed status to VERIFIED. So this bug has to be reopened later (in worse case, the bug has been left over and no one reopen it).

The second pattern shown in Fig 2 (b) called *Invalid/Duplicated-after-new* indicates that bugs have been detected as INVALID or DUPLICATE after they had been marked as NEW (e.g., NEW ⇒ ASSIGNED ⇒ RESOLVED (INVALID) or DUPLICATE). In this case, we suspects that a developer who marked NEW made a mistake. This bug is not *new* but was actually duplicated or was invalid (sometimes invalid mean it is not even a bug.). In this paper, we will use *Bad-status-pattern* to represent the two bug status patterns above. We suspect that the bug life cycle followed *Bad-status-pattern* might be longer compared to the bugs without such the bad pattern. Thus, these bugs waste more developers' time and efforts.

**FINDING.**

After extracting above patterns from every bugs in Bugzilla, we were able to identify 405 bugs that followed *Reopen-after-verified/closed* patterns with 289 bugs that has been marked as *VERIFIED* by committers. The other bugs were verified or closed by other developers that had verified permission but does not have commit permission. Thus, they are not committers. And for 696 bugs that followed Duplicate/Invalid-after-new patterns, there were 470 patterns that had been marked as new by our committers.

By using only bug reports that were involved with the committers (total of 52,013 bug reports), we used a box plot to compare the bug life cycle between the bugs that followed

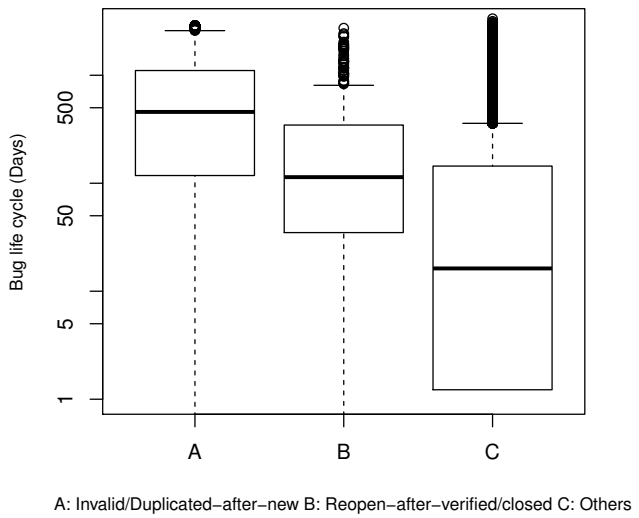



**Figure 3: Box plot comparing the bug's life cycle that followed *Invalid/Duplicated-after-new* patterns and *Reopen-after-verified/closed* patterns with normal bugs**

*Reopen-after-verified/closed* patterns, *Invalid/Duplicated-after-new* patterns , to other bugs that our committer has been involved. There were 51,254 bug reports that did not follow *Bad-status-pattern*. The results showed significant different number of days between these bugs. Unsurprisingly, the bugs where a committer made a mistake has longer bug life cycle.

# 5. DISCUSSION

In this section, we discuss the results from our research question and additional results we can find during our research.

## 5.1 Committer's Uncautious Activities: Negative Effects On The Bug Fixing Process

From our research question's results, we suspect some activities that potentially has a negative consequence and we compared it with normal activities. We can identify that the patch that has been *Reopen-after-committed* or the bugs that followed *Bad-status-pattern*s have longer life cycles compare to the other bugs. From this result, we wish to make a humbly suggestion to an OSS's committer to be aware of their importance to the bug fixing process. When they are not cautiously doing their jobs, they might extend the bug life cycles.

## 5.2 Additional Results: Not all Reopend Bug Are Bad

When we observed the bug status patterns in BTS, against popular beliefs, we have found that not all reopens have negative effects to the project. In the Eclipse-Platform, we can identified that there are 6 types of reopens. Each type has different impact to the bug life cycle. Some patterns showed that reopens can have positive effect such as ... ⇒ RESOLVED(WONTFIX) ⇒ REOPENED ⇒ ASSIGNED ⇒ RESOLVED(FIXED). We manually observed this patterns, found that some developers simply change the bug resolution to WONTFIX because they did not have enough knowledge to fix it, which later has been REOPENED and fixed by other developer. Another example is the reopened-after-later pattern ( ... ⇒ RESOLVED(LATER) ⇒ REOPENED ), this reopen is actually intended, LATER resolutions usually mean "this bug must wait for the new patch to be fixed", "this is not the target milestones", or "need some minor tweaks later". We want to make suggestion to other researchers who use bug status pattern in their work to be aware of these types of reopen and their different impacts.

# 6. LIMITATION AND FUTURE WORK

In our extraction process, we have collected each committer's patch commitment activities by the observed CVS description that had a link to the bug database. Unfortunately, this collected links number are considered to be small in portions compared to all of the activities in the CVS. From 30,833 commits in the commit log, we can identify only 1,193 links with a unique bug-id. To reduce the bias resulted from a sample size, our goal was to capture the largest representation of population as we can. As we explained earlier, we (hopefully) archived this goal by adapting [17] approach in order to overcome this limitations.

Please be noted that the results from this research is focus only on the Eclipse-Platform's development community. This community structure are well-organized and have full-time workers (like commercial development community). Different OSS communities can have different structure which will reflects in different results from the same approach.

In order to observe variation of the results reflected from the different communities, our future works will apply the approaches used in this research to another OSS projects and we would like to look deeper into existing committers themselves to find the committer candidates who will become "**good**" committers. We also would like to observe another developer's role in the OSS's bug fixing process, and hopefully received a useful results that can benefit all OSS communities.

# 7. CONCLUSION

In this paper, we have focused on developers who played a major role in bug fixing process called *Committers*. We suspected that, when the bugs are taken care by more cautious developers (and verified by cautious committer), their life cycles might be shorter. We identified committer's activities that have different consequences to the bug fixing process. Our findings can be summarized as follows:

- We ware able to determined the patches that have been reopened after it was committed and showed that, when the committer committed the patch and that patch had to be reopened later, it tends to have longer life cycle.
- We categorized severals bug status patterns, showed that when the bugs have its status followed *Bad-status-pattern*, they have longer bug life cycle than the bugs with other patterns.

## 8. ACKNOWLEDGMENT

The first author is grateful to the internship program cooperated and supported between Kasetsart University, Thailand, and Nara Institute of Science and Technology, JAPAN. It bestows a grant as well as an opportunity for undergraduate student to achieve a wealth experience in abroad graduated school research.

This work is also conducted as part of StagE Project (the Development of Next Generation IT Infrastructure), Grant-in-Aid for Scientific Research (B), 23300009, 2011, and Grant-in-aid for Young Scientists (B), 22700033, 2011 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

## 9. REFERENCES

[1] J. Anvik, L. Hiew, and G. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering (ICSE'06)*, pages 361–370, 2006.

[2] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *SIGSOFT'10/FSE-18: Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering*. ACM, 2010.

[3] N. Bettenburg, S. Just, A. Schröter, 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 (SIGSOFT'08/FSE-16)*, pages 308–318, 2008.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting structural information from bug reports. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 27–30, 2008.

[5] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and Balanced? Bias in Bug-Fix Datasets. In *Proceedings of the the Seventh joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2009.

[6] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work (CSCW'10)*, pages 301–310, 2010.

[7] S. Fujita, M. Ohira, A. Ihara, and K. ichi Matsumoto. An analysis of committers toward improving the patch review process in oss development. In *Supplementary Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE2010)*, pages 369–374, November 2010.

[8] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10) - Volume 1*, pages 495–504, 2010.

[9] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. "not my bug!" and other reasons for software bug report reassignments. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW'11)*, pages 395–404, 2011.

[10] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE'07)*, pages 34–43, 2007.

[11] A. Ihara, M. Ohira, and K. ichi Matsumoto. An analysis method for improvinga bug modification processin open source development. In *In 10th international workshop on principles of software evolution (IWPSE'09)*, pages 135 – 143. ACM, August 2009. Amsterdam,The Netherland.

[12] C. Jensen and W. Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering (ICSE'07)*, pages 364–374, 2007.

[13] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE'09)*, pages 111–120, 2009.

[14] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR'09)*, pages 131–140, 2009.

[15] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering (ICSE'07)*, pages 499–510, 2007.

[16] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. ichi Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *the 17th Working Conference on Reverse Engineering (WCRE 2010)*, pages 249–258. IEEE, IEEE Computer Society, October 2010.

[17] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the Second International Workshop on Mining Software Repositories*, pages 24–28, May 2005.

[18] 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*, pages 45–54, 2010.

[19] The Bugzilla Team. The Bugzilla Guide: 5.4. Life Cycle of a Bug. http://www.bugzilla.org/docs/3.4/en/html/Bugzilla-Guide.html.

[20] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering (ICSE'08)*, pages 461–470, 2008.