IEICE TRANS. INF. & SYST., VOL.E95–D, NO.9 SEPTEMBER 2012

PAPER Special Section on Software Reliability Engineering

# Good or Bad Committers? —— A Case Study of Committer's Activities on the Eclipse's Bug Fixing Process

**Anakorn JONGYINDEE**[†a)], ***Student Member***, **Masao OHIRA**[††b)], **Akinori IHARA**[†††c)], ***and*** **Ken-ichi MATSUMOTO**[†††d)], ***Members***

**SUMMARY** There are many roles to play in the bug fixing process in open source software development. A developer called "Committer", who has a permission to submit a patch into a software repository, plays a major role in this process and holds a key to the successfulness of the project. Despite the importance of committer's activities, we suspect that sometimes committers can make mistakes which have some consequences to the bug fixing process (e.g., reopened bugs after bug fixing). Our research focuses on studying the consequences of each committer's activities to this process. We collected each committer's historical data from the Eclipse-Platform's bug tracking system and version control system and evaluated their activities using bug status in the bug tracking system and commit log in the version control system. Then we looked deeper into each committer's characteristics to see the reasons why some committers tend to make mistakes more than the others.

***key words:*** *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 of software use and development. As OSS has become more common and popular among us, however, OSS projects are facing with a big challenge on their quality assurance activities. Due to the growing user base, especially large OSS projects such as the Mozilla and Eclipse projects, they have receives a considerable amount of bug reports from the users on a daily basis [1] (e.g., several hundred bug reports are posted to the Bugzilla [2] database of the Mozilla project every day). OSS projects require finding an effective way of dealing with a large number of bug reports. In an OSS project, a bug is fixed through the bug fixing process [3] which starts from the process where the bug is reported in the project until patches for fixing the bug are submitted into a software repository. Each bug report in this process is passed through one or more developers who play different roles before it is closed.

Manuscript received December 12, 2011.
Manuscript revised April 20, 2012.

[†]The author is with the Faculty of Computer Engineering, Kasetsart University, Bangkok, Thailand.
[††]The author is with the Faculty of Systems Engineering, Wakayama University, Wakayama-shi, 640–8510 Japan.
[†††]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
a) E-mail: b5105896@ku.ac.th
b) E-mail: masao@sys.wakayama-u.ac.jp
c) E-mail: akinori-i@is.naist.jp
d) E-mail: matumoto@is.naist.jp

DOI: 10.1587/transinf.E95.D.2202

In this study, we studied a developer who has a privilege to submit patches into the software repository, called *Committer*. This group of developers play major roles in the bug fixing process [4]. Their main task is to review (and sometimes edit) patches posted from other developers and then submit them into the software repository. Some of them also perform other tasks including bug resolution and bug reports management. Using a concurrent version system (CVS) and bug tracking system (BTS), they resolve bugs by themselves, join discussions about bugs, verify fixed bugs by developers, close bug reports, and so forth. As just described, committer's activities are vital for sustaining and improving the quality of OSS products.

However, committers are not always perfect. Bugs has been verified or closed in the past are occasionally reopened to be solved again because committers who verify or close the bugs might not cautiously review patches for bug fixing nor fully understand root causes generating the bugs in the initial trial for bug fixing. This can result in creating another bug report for the same bug again (i.e., reopen bug) and then wasting additional effort from already busy developers. These incidents also lead to "surprise defects [5]" that catch the software practitioners off-guard and disrupt the workflow of developers.

Eclipse-Platform is the large-scaled and well-known OSS project. The core developers of this project are employed by IBM and full-time workers for the project. They well-organize the project and dedicate to maintaining quality products. Researchers in the empirical software engineering literature sometimes regard the project as a proprietary software development project in a software company in terms of some common activities (e.g., quality assurance) [6]. Selecting Eclipse-Platform as a case study would lead to provide a useful insight on both OSS and commercial software development. At the same time, we believe that answering the following research question could contribute to improve and/or refine the results of previous studies [7]–[9] which had relied on the Eclipse-Platform's data set.

**RQ: What characteristics relates to the more cautious committers? And how about the lesser one?**

We suspected that committers who are more cautious should have different characteristics from the less-cautious ones. We classified their activities, based on the consequences to the bug fixing process such as the life cycle of

the bug, then evaluated and compared each committer using their activities count, as we attempt to find characteristics that separate better committers from the others.

By analyzing some committer's activities which can have a negative impact on the bug fixing process, in this paper we try to provide the following contributions:

- **Means to identify committers and their activities:** Using commit logs in CVS, we identify committers from thousands of developers in the Eclipse-Platform project. If a developer has committed a patch into CVS, s/he can be regarded as a committer. Then we collect and analyze their activities recorded on both CVS and BTS, using the "links" which will be further described in Sect. 3.
- **Better understandings of the consequences of committer's activities to the bug fixing process:** After we collect committers' activities, we look at the consequences to the bug fixing process in terms of the bug life cycle where we can assume that shorter time to solve a bug is more preferable for the bug fixing process. Analyzing the relationships between commuters' activities and time to resolve bugs would bring better understandings of which activities is more preferable or should be avoided for achieving an efficient bug fixing process.
- **Insights on characteristics that make committers more cautious:** Based on the analysis of committer's activities and their consequences, we classify committer's characteristics into four categories to find a way that makes committers more cautious in reviewing and/or fixing bugs. We would like to show that committers in the project should be aware of the importance of their roles and be cautious in doing their tasks in order to reduce "surprise defects" that effects to stakeholders.

In what follows, we introduce our related work and our motivation of this study in Sect. 2. Section 3 describes committer's activities and their consequences in the bug fixing process of OSS development. Section 4 introduces a data extraction method to answer the research question and Sect. 5 shows the analysis results. Additional interesting results that we were able to identify during this work are discussed further in Sect. 6. Section 7 describes limitations of this study, and we summarize our study in Sect. 8.

## 2. Related Work and Motivation

Most of existing studies are focusing 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 improve 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 to reduce the time to fix bugs because it can help developers to quickly find, replicate and understand the bugs at hand. However, developers' information needs in bug reports are often unsatisfied, since users do not know what information are required to fix a problem and so rarely articulate the problem on 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 [10]–[14] have interviewed with OSS developers and users to understand the information needs for bug fixing. For example, through interviews with over 150 developers and 300 reporters of the Apache, Eclipse and Mozilla projects, Bettenburg et al. [11] 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 which had been reported by another user in the past or which has already been fixed by developers. Sometimes developers also 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 is provided to find bugs reported in the past. In this manner, the same bugs are duplicated in BTS and then result in wasting developers' time and efforts.

To avoid duplicate bugs in BTS, several studies [15]–[18] have tried to detect duplicate bug reports automatically. For example, Wang et al. [18] 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 and then reassigned (*tossed* [1]) to other developers. This often happens because a triagger assigns 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 [1]. Preventing the bug tossing (assigning a bug fixing task to appropriate developers) is very effective to reduce the time to fix bugs.

Several approaches [1], [7], [19]–[24] exist in this

topic. For instance, Anvik et al. [19], [20] proposed an approach to assign a bug to an appropriate developer based on past bug reports with natural language processing. Jeong et al. [1] 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 involve in achieving better understandings on why reassignment occurs many times [23] and in creating a method to predict which bugs will be reopened or get fixed without being reopened [7], [24].

### 2.4 Cautious and Incautious Committers

In contrast with the previous studies above, in this paper we have studied the committer's 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 mistakenly review and accept a patch posted by other developers to fix a bug and then commit it into the repository that would bring reopen and another bug report. In this study we are interested in having a clear understanding of committer's activities and the consequences to the bug fixing process in order to emphasis the importance of their activities, avoid the incautious acts and hopefully reduce breakage and surprise defects in the project.

## 3. Committer's Activities and Consequences

This section describes how to capture committer's activities in the bug fixing process and how to find the consequences of the activities to the process.

### 3.1 Committer's Activities in the Bug Fixing Process

As described in Sect. 1, committers devote much effort to review (and sometimes edit) patches posted from other developers to the bug tracking system such as Bugzilla and then submit the patches into the version control system such as CVS. Some of them also perform other tasks including bug resolution and bug reports management on the bug tracking system. Therefore, committer's activities in the past are recorded in repositories of such the systems which are used by many researchers as rich information sources to test a hypothesis [4], create a prediction model [7], or performs statistical analysis [25].

In this paper we focus on committer's activities recorded in the bug tracking system especially since we are strongly interested in capturing consequences of their activities to the bug fixing process and then finding what characteristics of a committer contribute to more cautious about the bug fixing. We use 16 metrics which can be extracted from the repository data as shown in Table 1. The next section describes some of the metrics and why we need to see them.

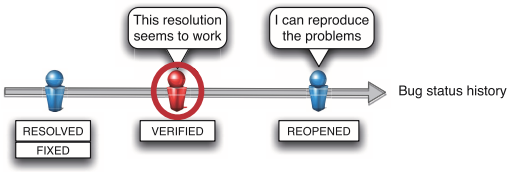
**Table 1** Metrics for measuring committer's activities.

| Metrics | Description |
|---|---|
| BSPR | *Bad-Status-Pattern* Rate: Number of *Reopen-after-verified/closed* and *Invalid/Duplicate-after-new* divided by the total number of status changes in BTS |
| RACR | *Reopen-after-committed* Rate: Number of bug that has been reopened after committed divided by the total number of commits |
| MTFB | Median Time that a committer takes to Fix Bugs |
| NASB | Number of Activities a committer Showed on BTS |
| TPJP | Time Period that a committer Joined a Project |
| NMCC | Number of Months a committer showed his Contributions as a Committer |
| TICC | Time Intervals from a latest bug status Change to a Commit for a bug fix |
| MBRT | Median Bug Review Time: Medial time until a committer decides to VERIFIED/CLOSED bugs |
| ABRT | Average Bug Review Time: Averatge time until a committer decides to VERIFIED/CLOSED bugs |
| NTSB | Number of Times a committer RESOLVED Bugs |
| NTAB | Number of Times a committer ASSIGNED Bugs |
| NTFB | Number of Times a committer FIXED Bugs |
| NTRB | Number of Times a committer REOPENED Bugs |
| NTVB | Number of Times a committer VERIFIED/CLOSED Bugs |
| NTNB | Number of Times a committer NEW Bugs |
| MTRB | Mean Time for Resolving Bugs |
| ATRB | Average Time for Resolving Bugs |

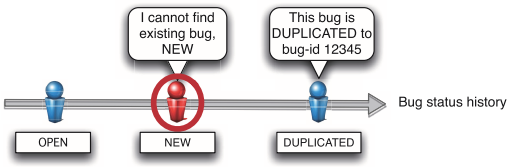
### 3.2 Consequences of the Activities to the Process

In Bugzilla, 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 of bug status have its own resolution such as FIXED, INVALID, and DUPLICATED [3] to indicate what happened to the bug. Using these information of the status changes, we can define *good* and/or *bad* consequences to the bug fixing process. In this paper when we describe the bug status that changed from one to the others in the bug history, for better clarifications we present the bug history in the form of bug status patterns. We use "⇒" for separation between bug status. The time dimension flows 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 exists. These bug status patterns 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 former pattern, we can observe that the bug has been assigned only once before it was resolved. This type of pattern usually leads to short or normal bug life cycle while the more complex one often leads to longer bug life cycle.

[†]http://www.bugzilla.org/docs/3.4/en/html/Bugzilla-Guide.html

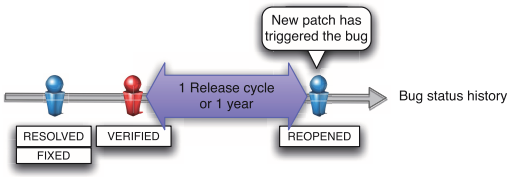



(a) *Reopen-after-verified/closed* pattern

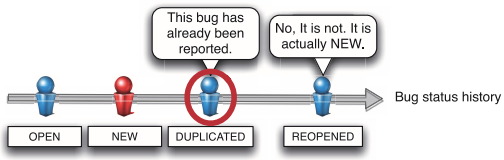



(b) *Invalid/Duplicated-after-new* pattern

**Fig. 1** The *Bad-status-patterns* observed in our study.




(a) *Reopen-after-verified/closed* pattern with late reopen




(b) Mistakenly marked as *Invalid/Duplicated* pattern

**Fig. 2** The other patterns excluded from our results.

#### 3.2.1 Bad Status Patterns

We can identify two status patterns that potentially have a negative effect on the bug fixing process we call *Bad-status-pattern*. In this work, we suspect that the bug life cycle followed *Bad-status-pattern* might be longer, compared to the bugs without such bad patterns. Thus, these bugs waste more developers' time and efforts. Observing the relationship between the bad status patterns and consequences (i.e., the delay of bug life cycle) would help us validate our assumption.

(1) Reopen-after-verified/closed

The first pattern shown in Fig. 1 (a) called *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 reopens it). To optimize our data quality, for the *Reopen-after-verified/closed* pattern, we had to exclude results if the time interval between VERIFIED or CLOSED status and REOPENED status is longer than a year or one release cycle. The reasons for this late reopen is usually because new patch released, new library introduced, or new class in this version which leads to bug reopen but not because the developers incautiously reviewed it and marked this bug as VERIFIED/CLOSED. Fig. 2 (a) has shown this pattern.

(2) Invalid/Duplicated-after-new

The second pattern is *Invalid/Duplicated-after-new* shown in Fig. 1 (b) which 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 makes a mistake. This bug is not *new* but actually duplicated or invalid (sometimes invalid means it is not even a bug.). We note that we also had to exclude some cases from this pattern. When a bug had been reopened (and sometimes fixed) after it was marked as INVALID or DUPLICATED (e.g., NEW ⇒ ASSIGNED ⇒ RESOLVED (INVALID/DUPLICATE) ⇒ REOPENED ⇒ RESOLVED (FIXED)), it would mean that there was a developer who marked the bug as a INVALID/DUPLICATED fault. S/he misunderstood that this bug is invalid or duplicated, which is actually not. For better clarifications, this pattern is illustrated in Fig. 2 (b).

(3) Reopen-after-committed

Another bad status pattern can be also captured by using both CVS and BTS data. If a bug report has been reopened after committer committed a patch posted by other developers or himself to CVS, we can consider it as the similar situation where the *Reopen-after-verified/closed* pattern occurred.

## 4. Analysis Method

In order to answer our research questions, we need to mine repository data from CVS and BTS as the following three steps: (1) identifying committers from an amount of developers, (2) finding the bad status patterns and (3) categorizing committers into four groups.

### 4.1 Identifying Committers

Before analysis we need to identify who are the committers, excluding them from thousand of regular developers in a project. To our knowledge, there is no specific activity that can decide whether one developer is a committer or not. [4] suggests only a rough descriptions how they extracted their committer list. They defined a developer as a committer who has a privilege to submit a patch to CVS. We also use the definition and can create a similar list of committers by scanning every line of commit logs and finding committers' names. After we get the committers' list, we can extract the

information on committer's activities from BTS.

### 4.2 Finding the Bad Status Patterns

As described earlier, the bad status patterns would have a negative impact on the bug fixing process. Using the status change information preserved in the BTS repository, the patterns *Reopen-after-verified/closed* and *Invalid/Duplicated-after-new* can be detected easily, but the *Reopen-after-verified/closed* pattern is a bit difficult to detect since we need to create links that shows which commits in CVS includes which patches posted to BTS.

Unfortunately we can only have a narrower vision of committer's activities from a commit log. It captures revision numbers, date of commit and some information about source code changes. The description about each change solely depends on committer's opinion. This description field has no centralized format and often is recorded in an ill-organized format (some of them are empty sometimes). Due to these inconsistencies, the CVS description itself is not adequate to judge each committer's behavior. In order to overcome this problem, we have decided to adapt T. Zimmerman's [26] approach to our study. Their approach suggests that despite its inconsistencies, sometimes committer mentions bug-id in the CVS description. By using bug-id as a trail, we can identify it as a *links* from the CVS repository to the BTS repository. From these *links* we can look further into the BTS repository where we have wider behavior information to study. The technique on finding these *links* has been used in many works (e.g., [8], [26], and [27] have described these links and illustrated it clearly.) in the field.

### 4.3 Categorizing Committers into Four Groups

We can measure committer's activities using the committer list. The 16 metrics to measure the activities would help us find which activities induce or avoid the bad status patterns. Furthermore, by categorizing committers into four groups, we would like to better understand the reasons why some committers are better than the others and which characteristics makes committers more cautious or less. We use the following procedure to categorize and evaluate committers.

1. Categorizing each committers into four groups depending on whether committers actively commit patches or not (i.e., whether committer's activities are frequently observed on CVS) and whether they actively review and/or modify bugs or not (i.e., whether committer's activities in BTS contribute to frequent bug status changes).
2. Evaluating each committer in each group based on their activity history (i.e., we evaluate him based on the number of the *Reopen-after-committed*, *Reopen-after-verified/closed* and *Invalid/Duplicated-after-new* patterns. The committer with the higher number of the bad status patterns is regarded as less cautious.)
3. Finding which characteristic(s) is related to the factor which makes a committer more or less cautious

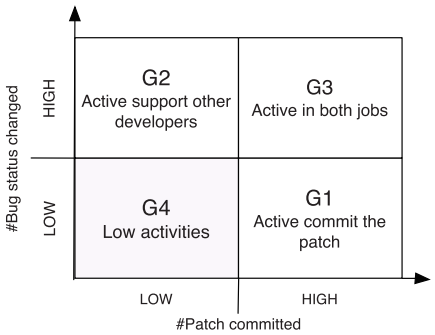



**Fig. 3** Categorization of committers. We used the median value of the histograms to create the threshold for categorizing them into each group.

Figure 3 illustrates how we categorize committers. We have noticed in collecting data from the Eclipse-Platform project that some committers devoted their effort to the patch commitment (e.g., some committers committed files to CVS over thousands times.) while some committers preferred to involve in fixing bugs on BTS. From this wide range of committer's activities, it would not be fair to evaluate them without having any normalization. That is the reason we categorized them into the four groups base on their activities. We collected each committer's activates based on the number of committed patches and the number of changed status in bug reports. Then we created two histograms: one is for the number of committed patches and another is for the number of changed status in bug reports. Then we used the median value in each histogram to define the threshold for categorizing into the four groups (14, 15, 22, and 23 committers in G1, G2, G3, and G4 respectively). We can consider committer's characteristics in each group as follows.

G1: This group of committers prefer to review and commit patches. They have larger number of patch commits, compared to the number of bug status changed. In order to evaluate a committer in this group we can use their *Reopen-after-committed* rate (RACR). Committers who cautiously review the patch before they commit would have lower rate than the one who does not.

G2: In contrast with G1, these committers have larger number of bug status changed, actively take care of bug reports for bug fixing. Because their smaller number of patch commits, solely counting their commit activities in CVS is not adequate to evaluate them. We evaluate each committer using the *Bad-Status-Pattern* rate (BSPR). Lower BSPR indicates that they are better committers.

G3: Some committers are very active in both bug fixing and patch commitment. For committers in the groups, we can use both of PACR and BSPR.

G4: These committers show no significant activities in both BTS and CVS. Their activities should be too low to be considered. As a result, we exclude results of this group from our analysis.

## 5. Results

This section presents the results of our case study of the Eclipse-Platform project.

### 5.1 Summery of Data Set

Using both bug reports in Bugzilla and commit logs in CVS which were created in the Eclipse-Platform project, we could collect 85,387 bug report data and over 30,833 commit log data. As a result, we captured activities of 2,584 different developers from October 2001 until January 2010. Table 2 shows the summary of our data set which was extracted according the extraction method described in the previous section.

**Table 2** Summary of data set extracted from Eclipse-Platform.

| Target period | 2001/10/01–2010/01/31 | |
|---|---|---|
| Commits | # of commits in total | 30,833 |
| | # of links between CVS and BTS | 1,193 |
| | # of *Reopen-after-committed* | 140 |
| Bug reports | # of bug reports in total | 85,387 |
| | # of bug reports involved with committers | 52,013 |
| | # of bug reports with *Reopen-after-verified/closed* | 405 |
| | # of bug reports with *Duplicate/Invalid-after-new* | 696 |
| Developers | # of developers in total | 2,584 |
| | # of committers (of 2,584 developers) | 74 |

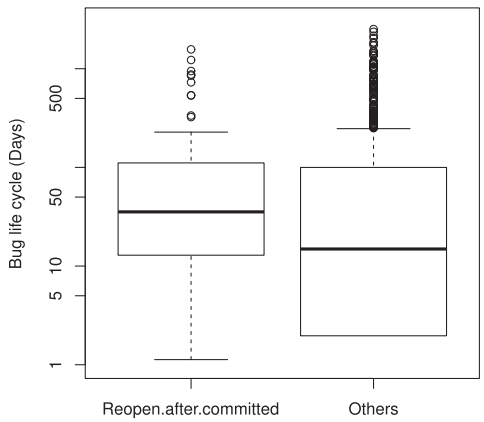



**Fig. 4** The life cycle of *Reopen-after-committed* and normal bugs.

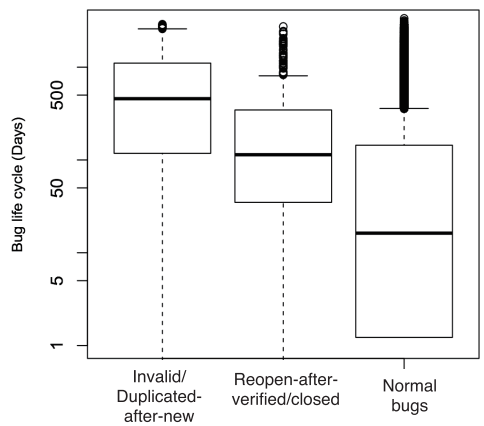



**Fig. 5** The life cycle of *Invalid/Duplicated-after-new*, *Reopen-after-verified/closed* and normal bugs.

Figure 4 shows one of the bad status patterns (*Reopen-after-committed*) tends to take longer time than other bugs until the bugs were resolved. As a result of Mann-Whitney's U test, we confirmed a significant difference ($p < 0.01$) between the two kinds of bug resolution time. We also did the same test for the rest of the patterns (*Reopen-after-verified/closed*, *Invalid/Duplicated-after-new, and normal bugs*) and confirmed a significant difference ($p < 0.01$) between them. We can say that the bug life cycle in Fig. 5 also have different number of days to resolve bugs. From these results, we can say that committer's incautious actions in the bug fixing process make bug resolutions slower and that the rate of *Bad-status-patterns* (i.e., RACR and BSPR) can be used as indexes for measuring committer's cautiousness to the bug fixing tasks.

### 5.2 *RQ: What characteristics relates to the more cautious committers? And how about the lesser one?*

After we evaluate committers in each group, we can answer our research question. We used the correlation-coefficient as a statistical tool to find a linear relationship between each committer's characteristics. We have compared each committer with the total of the 16 metrics.

*More experienced committer tends to submit cleaner patch*: From all metrics data we collected, number of months committer joined the project (NMCC) and the *Reopen-after-committed* rate (RACR) only showed a negative linear relationship with the correlation coefficient of $-0.68$. Interestingly, when they has been promoted to be a committer for longer period of time, they submitted patches which tends to be cleaner. In the other words, the more they experienced the more they would act cautiously.

## 6. Discussions

### 6.1 Committers who Mistakenly Verified/Closed Tend to Mistakenly Introduce a New Bug

After we answer our research question, we suspects that if committers have larger number of *Reopen-after-verified/closed* bugs, they might also have relatively larger number of *Invalid/Duplicated-after-new* bugs. In the other words, when committers *VERIFIED/CLOSED* bug negligently, they are likely to mark bug status as *NEW*. So we looked further into the group of committers who actively contributed to bug fixing tasks (G2). Figure 6 shows the box plot between these two patterns. We can confirm that larger number of false *VERIFIED/CLOSED* leads to larger number of false *NEW*.

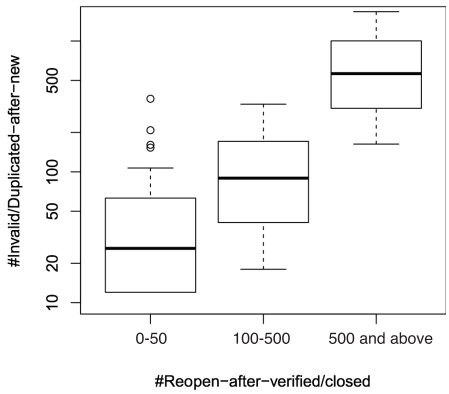



**Fig. 6** The relationship between *Reopen-after-verified/closed* and *Invalid/Duplicated-after-new* bugs.

**Table 3** Analysis results on the relationship between the bad status patterns and 16 metrics.

| | correlation coefficient | | | |
|---|---|---|---|---|
| | G1 | G2 | G3 | |
| Metrics | RACR | BSPR | RACR | BSPR |
| BSPR | −0.25 | N/A | −0.26 | N/A |
| RACR | N/A | −0.22 | N/A | −0.26 |
| MTFB | −0.14 | −0.21 | −0.45 | 0.34 |
| NASB | −0.35 | −0.12 | −0.26 | 0.13 |
| TPJP | −0.34 | 0.05 | −0.32 | 0.39 |
| NMCC | **−0.68** | −0.02 | −0.37 | 0.34 |
| TICC | 0.17 | −0.15 | 0.37 | −0.21 |
| MBRT | −0.22 | −0.10 | −0.20 | 0.17 |
| ABRT | 0.04 | −0.24 | −0.19 | 0.06 |
| NTRT | 0.08 | −0.11 | −0.26 | 0.14 |
| NTSB | −0.26 | −0.09 | −0.23 | 0.11 |
| NTAB | −0.29 | −0.13 | −0.24 | 0.11 |
| NTFB | −0.39 | −0.04 | −0.28 | 0.29 |
| NTRB | −0.26 | −0.16 | −0.16 | 0.04 |
| NTVB | −0.24 | −0.09 | −0.14 | 0.08 |
| MTNB | −0.05 | −0.09 | −0.08 | −0.11 |
| MTRB | 0.14 | −0.39 | −0.15 | −0.11 |

### 6.2 No Relationship between Mistakes in CVS and BTS

When committers are not cautious before deciding to commit a patch into CVS, they might not cautiously change bug status in BTS either. Interestingly, for the committers who were active in both tasks (G3), the relationship between their *Reopen-after-committed* rate (RACR) and *Bad-status-pattern* rate (BSPR) showed no significant linear relationship (correlation coefficient: −0.26). One explanation is that because of their wide range of activities, the committer's footprints in G3 are scattered all over the place, that is, their activities were distributed un-uniformly. More simple explanation is that when the committers decided to change bug status in BTS, they might use different kind of "cautiousness" from the patch commitment in CVS.

### 6.3 Not All Reopened Bug are Bad

When we observed the bug status change patterns in BTS, we have found that not all reopened bugs has a negative effect to the project, which is contradict to the popular belief. In the Eclipse-Platform project, we could identify six types of reopened bugs. Each type has different impact on the bug life cycle. Some patterns show that reopen can have a positive effect such as ...⇒ RESOLVED(WONTFIX)⇒ REOPENED⇒ ASSIGNED⇒ RESOLVED(FIXED). We manually observed these patterns and found that some developers simply changed the bug resolution to WONTFIX because they did not have enough knowledge to fix it and then the bug has been REOPENED later and FIXED by other developers. Other example is that the reopened-after-later pattern (...⇒ RESOLVED(LATER)⇒ REOPENED) were actually intended. LATER resolutions usually mean that the bugs must wait for the new patch to be fixed, that they were not the target milestones or that they needed some minor tweaks later. We would like to suggest that it is important for both the researchers and practitioners to be aware of these several types of reopen and their difference impacts.

## 7. Limitations

### 7.1 Activities in CVS

In our extraction process, we have collected each committer's patch commitment activities by observing CVS descriptions that has links to the bug database. Unfortunately, the current method can only extract small portions of links, compared to all of the activities in CVS. From 30,833 commits in commit log, we could identify only 1,193 links with unique bug-ids. To reduce the bias resulted from the small sample size, our goal was to capture as the largest representative of the population. As we explained earlier, we (hopefully) archived this goal by adapting the Zimmermann's approach [26] in order to overcome this limitation.

### 7.2 Interpretations of the Bug Status Patterns

We acknowledge that observing the developer's tasks solely based on the bug status patterns also has some limitations. One status pattern can have several meanings or can be interpreted in many ways such as the ...⇒ REOPENED⇒ ASSIGNED⇒ NEW⇒ RESOLVED(INVALID) patterns. For instance, let A be a developer who reopens this bug and let B be another developer who changes its status to RESOLVED and add INVALID to its resolution. We can interpret this pattern in two ways.

First, when A decided that this bug has to be reopened (i.e., due to new patch, class, library or other reasons), A assigned this bug to B. Later B found out that it is invalid (i.e., not a bug, false reproduce, etc.). In this case, it is an A's fault. For the second case, if this bug has never been fixed before and A decided to reopen this bug, assigned it to B, asked B whether this bug is invalid or not. Then, B helped A confirm that this bug is really invalid. Thus A has helped shorten the bug's life cycle.

In order to optimize our result accuracy, we have normalized the data by using only the bugs with same priority, assuming that developers are the bug tracking system properly. After randomly selecting some bugs to observe their status patterns manually, we satisfactorily found that most of the patterns used in this study were straight forward. We could easily identify a developer who mistakenly verified bugs, using the *Reopen-after-verified/closed* and the *Invalid/Duplicated-after-new* patterns respectively.

However, there might be more activities that lead to the same results which we have to include in our future works. For the committer's characteristics (i.e., the 16 metrics) we observed, we also noticed that there might be more things that we did not observe in this study. Please noted that the results from this study is focusing only on the Eclipse-Platform development community. This community is well-organized and have full-time workers from IBM. Other OSS communities can have difference structures which might derive difference results from the same approach.

## 8. Conclusion and Future Work

In this paper, we have focused on a developer who plays major role in bug fixing process called *Committer*. We have suspected that when the bugs are taken care by more cautious committers (and verified by cautious committers), their life cycles tends to be shorter. We identified committer's activities that have different consequences to the bug fixing process and categorized each committer based on their activities and then found the related characteristics that separated good committers from bad ones. Our findings can be summarized as follows.

- We were able to determine the patches that have been reopened after it was committed. When committers committed the patch and the patch had to be reopened later, it tends to have the longer bug life cycle.
- We identified several bug status patterns which present that when bugs follow the *Bad-status-patterns*, they have the longer bug life cycle than other patterns.
- From a wide range of committer's activities, we categorized committers into four types of groups.
- Based on the categorization, we could identify that committers who are active in CVS have more experienced. Such the committers tend to commit cleaner patches with the lower *Reopen-after-committed rate* (RACR).

Based on these findings above, we could answer our research question "*What characteristics relates to the more cautious committers? And how about the lesser one?*" in this paper: more cautious committers tend to be more experienced developers who are very active in committing cleaner patches rather than reporting bugs. In contrast, less cautious committers tend to commit less quality patches and bug reports that would cause the bad status patterns of bug reports (i.e., reopen after patch committed, verified, and closed).

We believe that our findings through a case study of Eclipse-Platform do not only provide a useful insight for both OSS and commercial software developers who dedicate to software maintenance and/or quality assurance activities, but also contribute to improve and/or refine software development support methods which had been proposed in existing studies (e.g., [1], [20], [28]).

In order to observe a variety of results from different OSS communities, our future work needs to apply the approach used in this study to another OSS projects. We would like to find other characteristics which makes committers more cautious. We also would like to observe another developer's role in the bug fixing process and hopefully receive useful results that would benefit open source development.

## Acknowledgment

This research is conducted as part of Grant-in-Aid for Scientific Research (B) 23300009 and (C) 24500041 by Japan Society for the Promotion of Science (JSPS).

### References

[1] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," Proc. 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE'09), pp.111–120, 2009.
[2] Bugzilla, "Bug tracking system." http://www.bugzilla.org/
[3] 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
[4] S. Fujita, M. Ohira, A. Ihara, and K. ichi Matsumoto, "An analysis of committers toward improving the patch review process in oss development," Supplementary Proc. 21st IEEE International Symposium on Software Reliability Engineering (ISSRE2010), pp.369–374, Nov. 2010.
[5] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A.E. Hassan, "High-impact defects: a study of breakage and surprise defects," Proc. 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11), pp.300–310, 2011.
[6] J. Businge, A. Serebrenik, and M. van den Brand, "An empirical study of the evolution of eclipse third-party plug-ins," Proc. Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), pp.63–72, 2010.
[7] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," 17th Working Conference on Reverse Engineering (WCRE'10), pp.249–258, 2010.
[8] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," Proc. 7th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE'09), pp.121–130, 2009.
[9] E. Giger, M. Pinzger, and H.C. Gall, "Comparing fine-grained source code changes and code churn for bug prediction," Proc. 8th Working Conference on Mining Software Repositories (MSR'11), pp.83–92, 2011.
[10] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," Proc. 2007 OOPSLA workshop on eclipse technology eXchange (Eclipse'07), pp.21–25, 2007.
[11] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T.

Zimmermann, "What makes a good bug report?," Proc. 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT'08/FSE-16), pp.308–318, 2008.
[12] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," Proc. 2008 International Working Conference on Mining Software Repositories, pp.27–30, 2008.
[13] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," Proc. 2010 ACM Conference on Computer Supported Cooperative Work (CSCW'10), pp.301–310, 2010.
[14] P. Hooimeijer and W. Weimer, "Modeling bug report quality," Proc. twenty-second IEEE/ACM International Conference on Automated Software Engineering (ASE'07), pp.34–43, 2007.
[15] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?," Proc. 24th IEEE International Conference on Software Maintenance (ICSM'08), pp.337–345, 2008.
[16] C. Sun, D. Lo, X. Wang, J. Jiang, and S.C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," Proc. 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10) - vol.1, pp.45–54, 2010.
[17] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," Proc. 29th International Conference on Software Engineering (ICSE'07), pp.499–510, 2007.
[18] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," Proc. 30th International Conference on Software Engineering (ICSE'08), pp.461–470, 2008.
[19] J. Anvik, L. Hiew, and G. Murphy, "Coping with an open bug repository," Proc. 2005 OOPSLA Workshop on Eclipse Technology eXchange (eclipse'05), pp.35–39, 2005.
[20] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?," Proc. 28th International Conference on Software Engineering (ICSE'06), pp.361–370, 2006.
[21] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," Proc. 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR'09), pp.131–140, 2009.
[22] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," Proc. 2010 IEEE International Conference on Software Maintenance (ICSM'10), pp.1–10, 2010.
[23] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""not my bug!" and other reasons for software bug report reassignments," Proc. ACM 2011 Conference on Computer Supported Cooperative Work (CSCW'11), pp.395–404, 2011.
[24] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," Proc. 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10) - vol.1, pp.495–504, 2010.
[25] A. Ihara, M. Ohira, and K. ichi Matsumoto, "An analysis method for improvinga bug modification processin open source development," In 10th international workshop on principles of software evolution (IWPSE'09), pp.135–143, Amsterdam, The Netherland, Aug. 2009.
[26] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," Proc. Second International Workshop on Mining Software Repositories, pp.24–28, May 2005.
[27] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: Bugs and bug-fix commits," SIGSOFT'10/FSE-18: Proc. 16th ACM SIGSOFT Symposium on Foundations of Software Engineering, 2010.
[28] S. Kim, T. Zimmermann, K. Pan, and E.J.J. Whitehead, "Automatic identification of bug-introducing changes," Proc. 21st International Conference on Automated Software Engineering (ASE'06), pp.81–90, 2006.

**Anakorn Jongyindee** is an undergraduate student from Computer Engineering, Kasetsart University, Thailand. He participated in an 8 weeks internship program between Kasetsart University and Nara Institute of Science and Technology in Japan from March to April of 2011, with a goal to study and experience conducting a research in Software Engineering field. His research interests include mining software repository and open source software analysis.

**Masao Ohira** received the B.E.degree from Kyoto Institute of Technology, Japan in 1998, M.E. and PhD degrees from Nara Institute of Science and Technology, Japan in 2000 and 2003 respectively. Dr. Ohira is currently Associate Professor at Wakayama University, Japan. He is interested in supporting knowledge collaboration in software development and supporting use and development of open source software. He is a member of ACM, Human Interface Society, and IPSJ.

**Akinori Ihara** received the BE degree in Science and Technology from Ryukoku University, Japan in 2007, and the ME degree (2009) and DE degree (2012) in information science from Nara Institute of Science and Technology, Japan. He is currently Assistant Professor at Nara Institute of Science and Technology. His research interests include the quantitative evaluation of open source software development process. He is a member of the IEEE and IPSJ.

**Ken-ichi Matsumoto** received the B.E., M.E., and PhD degrees in Information and Computer sciences from Osaka University, Japan, in 1985, 1987, 1990, respectivery. Dr. Matsumoto is currently a professor in the Graduate School of Information Science at Nara Institute Science and Technology, Japan. His research interests include software measurement and software process. He is a senior member of the IEEE, and a member of the ACM and IPSJ.