2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement

Good or Bad Committers? A Case Study of Committers' Cautiousness and the Consequences on the Bug Fixing Process in the Eclipse Project

Anakorn Jongyindee*, Masao Ohira†, Akinori Ihara†, and Ken-ichi Matsumoto†
*Faculty of Computer Engineering, Kasetsart University, Bangkok, Thailand
†Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan
Email: *b5105896@ku.ac.th, †{akinori-i, masao, matumoto}@is.naist.jp

***Abstract*—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 software repository, plays a major role in this process and hold a key to the successfulness of the project. In this work, we have observed each committer 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 mistakes, which have negative consequences to the bug fixing process. Our research focus on studying the consequences of each committer's activities to this process. We collected each committer's historical data and evaluated each of them by comparing the more cautiousness to less cautiousness committers. Then we looked deeper into each committer's characteristics to see the reasons why some committers tend to make mistakes more than the others. From our results, we would like to make a humbly suggestion to the OSS's committers to be aware of their importance to the projects and be cautious before doing their jobs.**

***Keywords*-open source software (OSS); committer; bug fixing process**

I. Introduction

Nowadays, 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. For instance, the Google's Android operating system has become the best-selling smart phone platform. Currently, OSS products have a large impact not only on the end-users but also the company users like manufacturers of mobile devices, since they need to exploit OSS to produce their end products.

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 starting from the process where the bug is reported in the project until patches for fixing the bug has 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 it is closed.

In this study, we focus on 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 (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, 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 uncautiously verify a bug report resolved by a developer and close the bug report, but this uncautious act can result in creating another bug report for the same bug again (i.e., reopen bug). In this paper, we are interested in creating a clear understanding on committers' cautiousness and their consequences on the bug fixing process, in order to find a better way to improve the bug fixing process in OSS projects.

A. Research Questions

Selecting Eclipse-Platform's version archives (CVS) and bug tracking database (BTS) as the information source of our case study, in this paper we ask the following research questions.

RQ1: What are the consequences of committer's cautiousness to the bug fixing process? In this question, we study the committer's cautiousness and its effects to the bug fixing process. Then we compared the consequence of more cautious activities to the lesser one.

RQ2: What characteristics relates to the more cautious committers? And how about the lesser one? With this question, we suspect that committers who are more cautious should have different characteristics from the less-cautious

978-0-7695-4565-3/11 $26.00 © 2011 IEEE
DOI 10.1109/IWSM-MENSURA.2011.24

ones. We classify, evaluate and compare each committer, as we attempt to find characteristics that separate better committers from the others.

B. Contributions

From the above research questions, we can provide contributions in this paper as follows.

- By understanding that some un-cautious committer activities can have bad consequences on the bug fixing, we would like to suggest OSS's committers to be aware of the importance of their roles and to be cautious in doing their tasks.
- From the seconds RQ, we will focus on studying the reasons why some committer acts are more cautious or uncautious than the others. We would know what is demanded to be a better committer and to avoid the incidents that make them less-cautious than the others.

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

II. 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.

A. 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 [5]–[9] 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. [6] have found that steps to reproduce and stack traces are most useful in bug reports.

B. 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 [10]–[13] have tried to detect duplicate bug reports automatically. For example, Wang et al. [13] present an approach to detect duplicate bugs based on a natural language processing techniques.

C. 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], [14]–[20] exist in this topic. For instance, Anvik et al. [14], [15] 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 established 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 [18] and in creating a method to predict which bugs will be reopened or get fixed without being reopened [19], [20].

D. Cautious and uncautious committers

In contrast with the previous studies above, we are interested in constructing a method to select committer candidates from existing developers in an OSS project because we believe that OSS projects should have more committers to handle a huge amount of bug reports. While the previous studies basically tried to reduce inefficient efforts in the bug fixing process, the aim of our study is to enlarge the ability of fixing bugs.

In general, a dedicated developer is nominated or elected to promote to a committer in an OSS project [21]. The OSS project carefully select 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 wishes 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 in a year and OSS projects are always facing with the difficulty in having more committers who greatly contribute to the bug fixing process.

In order to find a way to increase committers in OSS projects, Fujita et al. [4] 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 [4], in this paper we would like to look deeper into existing committers themselves to find committer candidates who will become "**good**" committers. 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 verify a fixed bug by developers, that would result in creating reopened bug reports in the future. They also might uncautiously review and accept a patch posted by developers to fix a bug and then commit 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 committers' cautiousness and the consequence on the bug fixing process and also interested in separating cautious committers from less-cautious ones to predict good committer candidates from developers.

III. EXTRACTION METHOD

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

A. Committers' activities in the Eclipse-Platform project

When developers are involved in the bug fixing process, their actions 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" [3]). Developers in the project can see the information of the database in the HTML form through a web browser.

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

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

In order to answer our research questions, first, we need to identify who are the committers, 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. [4] suggests only a rough description in how they extract their committer list. Based from their work, they have 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, his action is captured and his 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 have 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 collect their patch commitment footprint left on CVS data.

B. Observing committer's support activities from BTS

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 of bug status have its own resolution to indicate what happened to the bug such as FIXED, INVALID, and DUPLICATED [3].

Bug status history are used in many researches as a very useful source of information. Researchers can test a hypothesis [4], create prediction models [22], or performs statistical analysis [23]. In this paper, when we describe the bug status that changed from one to the others in the bug history, for better clarifications, we presents the bug history in the form of bug status patterns. We use "⇒" for separation

[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.

between bug status. Time dimension is 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 exists. 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 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 involved by our committers with 4,941 of 30,833 difference bug status patterns.

C. Observing committer's main activities from CVS

For Eclipse-Platform's commit log, we wish we could observe committer's main behaviors solely from the commit log as we did in Bugzilla. Unfortunately from this commit log we can only have a narrower vision of committer's activities. It captures revision numbers, date of commit and some information about source code changes. The description about each change is solely depends 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 have decided to adapt T. Zimmerman's [24] approach to our study. Their approach suggested that, despite its inconsistencies, sometimes committer mentions 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., [25], [24] and [26] 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 can collect each committer's main activities.

IV. RESULTS

A. RQ1: What are the consequences of committer's cautiousness to the bug fixing process?

We focus on the consequences of committer's activities. We separate 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.

1) Consequences of committer's main activities:

APPROACH: As mentioned above, we suspect that when a committer uncautiously committed the patch that fixed the bug, this bug might be reopened later to be resolved again. After compared 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 which had been reopened after committers committed the patches.

FINDING: According to the result shown in Fig 1, we can see that when a bug reopens after patches were committed, the bugs tend to have longer life cycles.

2) Consequences of committer's 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 have randomly chosen these patterns to inspect manually. By focusing only patterns related to

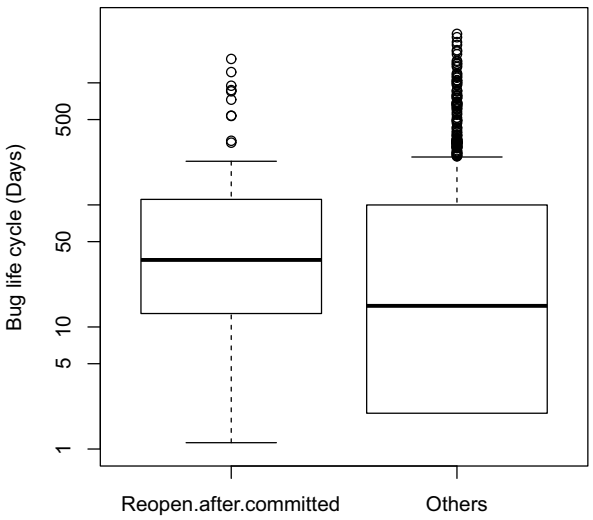


Figure 1. Box plot comparing the life cycle of *Reopen-after-committed* bug and normal bug

committer's jobs, we can identify two status patterns that potentially have a negative effect on the bug fixing process.

The first pattern shown in Fig 2 (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 reopen it).

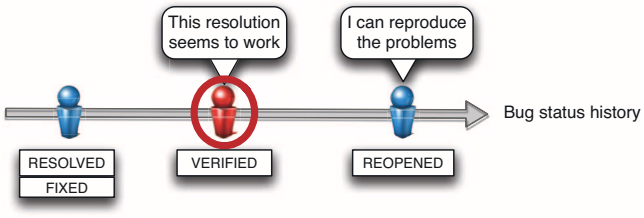
The second pattern shown in Fig 2 (b) called *Invalid/Duplicated-after-new* indicates that bugs have been detected as INVALID or DUPLICATE after they had been marked as NEW (e.g., NEW ⇒ ASSIGNED ⇒ RESOLVED (INVALID) or DUPLICATE). In this case, we suspects that a developer who marked NEW makes a mistake. This bug is not *new* but actually duplicated or invalid (sometimes invalid means 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 bad patterns. Thus, these bugs waste more developers' time and efforts.

While we manually observed the consequences of each *Bad-status-pattern*, we have noticed that there are some special cases to be considered. 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 uncautiously reviewed it and marked this bug as VERIFIED/CLOSED. Fig 3 (a) has shown this pattern.

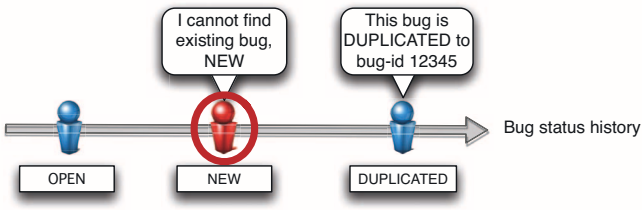
We also manually observed the next pattern called the *Invalid/Duplicated-after-new* pattern. We noticed some special pattern that we had to exclude from our result. We observed that when a bug had been reopened (and sometimes fixed) after it was marked as *INVALID* or *DUPLICATED* (e.g., NEW ⇒ ASSIGNED ⇒ INVALID/DUPLICATE ⇒ REOPENED ⇒ RESOLVED (FIXED)), it is a developer who marks this bug as *INVALID/DUPLICATED* fault. They misunderstood that this bug is invalid or duplicated, which is actually not. For better clarifications, this pattern is illustrated in Fig 3 (b).

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 marked as *VERIFIED* by committers. The other bugs are verified or closed by other developers who have 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 are involved with the committers (total of 52,013 bug reports), we use a box plot to compare the bug life cycle between the bugs that followed *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 which does not follow *Bad-status-pattern*. Fig 4 shows the results with significant difference number of days between these bugs. Unsurprisingly, the bugs where a committer made a mistake has longer bug life cycle.

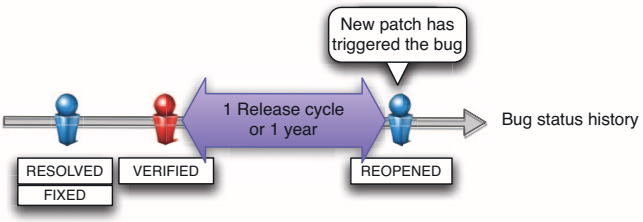


(a) *Reopen-after-verified/closed* pattern

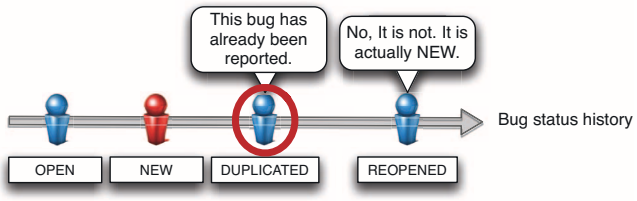


(b) *Duplicate-after-new* pattern

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



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



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

Figure 3. The other patterns excluded from our results

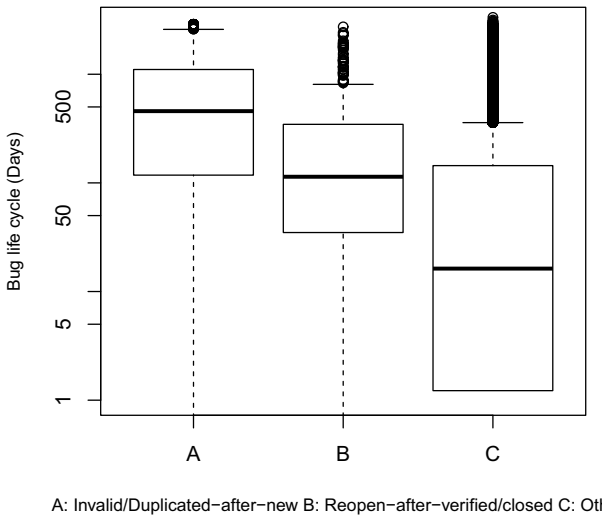


Figure 4. Box plot comparing the bug's life cycle that followed *Invalid/Duplicated-after-new* patterns and *Reopen-after-verified/closed* patterns with normal bugs

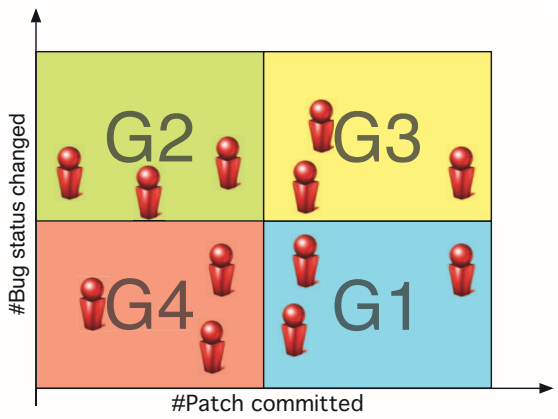


Figure 5. Show how we categorize each committer's type based on their activities count. Please be noted that, in order to separate the committer in each groups, we use histograms combining with some manual adjustment to create the threshold for categorize the committers.

B. RQ 2: What characteristics are related to more cautiousness committer? And what about the lesser ones?

APPROACH: In order to find the reasons (related characteristics) why some committers are better than the others, we use the following process to answer this research question:

- Categorize each committers into 4 groups based on their activities count. Group of committers who actively commit the patches, actively modify the bugs in BTS, Both active, and Low activity.
- Evaluate a committer in each groups based on their activities history (i.e., For the committer who actively commit the patches, we evaluate the committers in this group from more to less cautious by using their number of *Reopen-after-committed* patches, the higher the number the less cautious)
- We look further into each of these evaluated committer. Try to find which characteristic(s) is relate to the committer who is more cautious and which characteristic(s) often leads to less cautious committer.

While we are collecting each committer history, we can identify that there are wide range in number of BTS and CVS activities. We have noticed that some committers have contribute themselves to the patch commitment and have CVS activity count over thousands. While some committers had preferred to involve in BTS's bugs only. From this wide range of activity, it would not be fair to evaluate them without normalize. So, we categorize them into 4 groups base on their activity count. Fig 5 illustrates how we categorize them.

1) Committer who actively commit the patches (G1): Some committer only prefer to review and commit a patche into repository. This type of committers have relatively high number of patch committed compared to the number of bug status changed. (They actively do their main jobs.) In order to evaluate a committer in this group we use their *Reopen-after-committed* rate, calculated by counting number of *Reopen-after-committed* patches divided by number of patch commitments. Committer? who cautiously review the patch before they commit would have lower rate than the one who does not.

2) Committer who actively support other developers (G2): In contrast with above group. Some committer have high number of bug status changed, they actively doing their support jobs. With their low commitment count, their commit activities in CVS solely is un-adequate to judge them. So, we evaluate each committer using a *Bad-status-pattern* rate (The number of *Bad-status-pattern* divided by number of others patterns in BTS.) The lower the rate, the better committer they are.

3) Committer who active in both jobs (G3): Some committer are very active in both fixed a bugs and committed the patch. For committer in this groups, we can use both of their activities history to judge them.

4) Low activity committer (G4): These committers show no significant activities in both BTS and CVS. Their activities are too low to be considered. As a result, we have to exclude the results of this group from our analysis.

FINDING: After we evaluate the committer in each group, we can answer our research question. We use correlation-coefficient as a statistical tool to find a linear relationship between each committer's characteristics to the more cautious or less cautious committer in each group. We

Table I
RESULTS OF RQ2

Metrics	Description	correlation coefficient			
		G1	G2	G3	
		Reopen after committed	Bad pattern	Reopen after committed	Bad pattern
BR	Bad-pattern rate, number of Reopen-after-verified/closed and Invalid/Duplicate-after-new divide by support activities count	-0.25	1.00	-0.26	1.00
RAC	Reopen-after-committed rate, number of bug that has been Reopen-after-committed divide by number of links found.	1.00	-0.22	1.00	-0.26
MB	Median value of each committer's bug life cycle	-0.14	-0.21	-0.45	0.34
SA	Number of activities s/he shown in a Bug Tracking System	-0.35	-0.12	-0.26	0.13
PJ	Period of time s/he has joining the projects	-0.34	0.05	-0.32	0.39
CM	Number of months s/he show activities as a committer	-0.68	-0.02	-0.37	0.34
CDT	Time intervals between latest bug status before s/he decide to commit it to Commit log	0.17	-0.15	0.37	-0.21
MRT	Median value of Bug Review Time, Time before he decided to VERIFIED/CLOSED the bug	-0.22	-0.10	-0.20	0.17
ART	Average of Bug Review Time, Time before he decided to VERIFIED/CLOSED the bu g	0.04	-0.24	-0.19	0.06
RSC	Number of time s/he RESOLVED the bug	0.08	-0.11	-0.26	0.14
AC	Number of time s/he ASSIGNED the bug	-0.26	-0.09	-0.23	0.11
FC	Number of time s/he FIXED the bug	-0.29	-0.13	-0.24	0.11
ROC	Number of time s/he REOPENED the bug	-0.39	-0.04	-0.28	0.29
VC	Number of time s/he VERIFIED/CLOSED the bug	-0.26	-0.16	-0.16	0.04
NC	Number of time s/he NEW the bug	-0.24	-0.09	-0.14	0.08
MRST	Mean value of bug resolving time	-0.05	-0.09	-0.08	-0.11
ARST	Average value of bug resolving time	0.14	-0.39	-0.15	-0.11

suspects many characteristics that might be involved with each committer's behavior results such as more cautious committer might have a longer review time, or when some committers have more patches committed, they might leads to lower *Reopen-after-committed* rate. In this paper, we have compared each committer with the total of 16 characteristics, description about each characteristics and results are shown in Table 1.

V. DISCUSSION

In this section, we discussed with a results from our research questions and additional results we can find during our research.

A. Findings

Some committer's un-cautious activity can have negative effects to the bug fixing process: From our first research question's results, we suspects some activities that potentially has a negative consequence and we compared it with normal activities. We can identified that the patch that has been *Reopen-after-committed* or the bugs that followed *Bad-status-pattern*s are have longer life cycles compare to the others bug. From this results, 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, its might extend the bug life cycles.

More experienced committer tends to submit cleaner patch.: For G1, committer who active in CVS, as shown in Table 1, The CM Metrics, number of committer months, and *Reopen-after-committed* rate show significant negative linear relationship with correlation coefficient of -0.68. From this results we could see that, interestingly, when s/he has been promoted to be a committer for longer period of time, her/his submitted patches are tends to be cleaner. In the other words, the more s/he experienced the more s/he act cautiously.

B. Additional Results

Additional from the results from our RQ, we shown our other findings during our research in the following

Committer who mistakenly verified/closed tends to mistakenly new the bug: After we answer our research questions, we suspects that if the committer have high number of *Reopen-after-verified/closed* bugs, they might have relatively high number of *Invalid/Duplicated-after-new* bugs. In the other words, when committer's negligently *VERIFIED/CLOSED* bug, they are likely to negligently marked the bug status as *NEW*. So we look further into the committer who actively support other developers group (G2), Fig 6 show the box plot between these two patterns. We can determined that high number of false *VERIFIED/CLOSED* are leads to high number false *NEW*.

No relationship between CVS mistake and BTS mistake: From our second research question, we suspects that if committer not cautious before decided to commit the patch in CVS, they might not cautiously change bug status in BTS

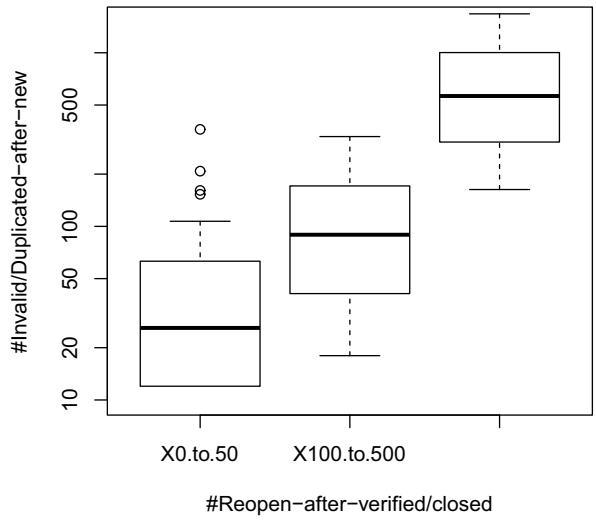


Figure 6. Box plot show the relationship between number of *Reopen-after-verified/closed* bugs and *Invalid/Duplicated-after-new* bugs.

either. Interestingly, From the committer who active in both jobs (G3 from the Table 1), after observed the relationship between their *Reopen-after-committed* rate and *Bad-status-pattern* rate (Metric RAC and BR respectively). The result show no significant linear relation between two values with correlation coefficient of -0.26. One explanation is that because their wide range of activities, Committer's footprint are found scattered all over the place, their activities count are distributed un-uniformly. More simple explanation is when a committer decided to change bug status in BTS, committer use difference kind of "cautiousness" from the patch commitment in CVS.

Not all reopen are bad: When we observe the bug status patterns in BTS, contradict to popular belief, we have found that not all reopen has negative effect to the project. In Eclipse-Platform, we can identified that there are 6 types of reopen. Each type has difference impact to the bug life cycle. Some patterns show that reopen 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 do not have enough knowledge to fix it. Which later has been REOPENED and fixed by other developer. Other example is the reopened-after-later pattern (...⇒ RESOLVED(LATER)⇒ REOPENED), this reopen are 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 another researchers who use bug status pattern in their work to be aware of these several types of reopen and their difference impacts.

VI. LIMITATIONS

CVS activities data limitations: In our extraction process, we have collected each committer's patch commitment activities by observed CVS description that has a link to the bug database. Unfortunately, this collected links number are considered to be small portions compared to all of the activities in CVS. From 30,833 commits in 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 is to capture as largest representation of population as we can. As we explained earlier, we (hopefully) archives this goal by adapting [24] approach in order to overcome this limitations.

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

First, If A decided that this bug has to be reopened (i.e., due to new patch,class,library or other reasons) , assigned this bug to B. Later B has 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 to B, asked B whether this bug is invalid or not. Then, B helps 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 randomly selected some bugs to observe their status pattern manually. We satisfactorily found that most of the patterns used in this research are straight forward, we can easily identify a developer who mistakenly verified or new the bug from the *Reopen-after-verified/closed* and the *Invalid/Duplicated-after-new* patterns respectively.

In RQ1, we observe only activities that we suspects its potentially have negative effect to the bug fixing process, there might be more activities that leads to the same results which we will include it in our future works. For the committer's characteristics we observe in Table 1. We also noticed that there might be more committer's activities that we did not observed, which can leads to more results either.

Please be noted that results from this research is focus only on Eclipse-Platform development community. This community structure are well-organized and have full-time workers. Difference OSS community can have difference

structural which will reflects difference result from the same approach

VII. CONCLUSION

In this paper, we have focus on a developer who plays major role in bug fixing process called *Committer*. We have suspects that, when the bugs are taken care by more cautious developer (and verified by cautious committer), their life cycles might be shorter. We identified committer's activities that have difference consequences to the bug fixing process, categorize each committer based on their behavior and then find a related characteristics that separated a good committer over the bad ones. Our findings can be summarized as follows

- We are able to determined the patches that have been reopen after it was committed and showing that, when the committer committed the patch and that patch had to be reopened later, its tends to have longer life cycle.
- We categorize severals bug status patterns, show that when the bugs have its status followed *Bad-status-pattern*, they have longer bug life cycle than the bugs with other patterns.
- From wide range of their activities, we can identified 4 types of committer based on their behavior count. Active in CVS, in BTS, Both active and Low active.
- We can identified that, for committer who active in CVS, more experienced committers tends to commit cleaner patches, with lower *Reopen-after-committed rate*.

In order to observe variation of the results reflected from the difference community, our future works will apply the approaches used in this research to another OSS projects, finding more characteristics that related to each type of committer, find the potential developer's characteristics that can leads to more cautious committer. We also would like to observe another developer's role in the OSS's bug fixing process, and hopefully received a useful results that would benefit all OSS community.

VIII. 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 research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 22700033, 2010 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

REFERENCES

[1] 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)*, 2009, pp. 111–120.

[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," in *Supplementary Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering (ISSRE2010)*, November 2010, pp. 369–374.

[5] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," in *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange (Eclipse'07)*, 2007, pp. 21–25.

[6] 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)*, 2008, pp. 308–318.

[7] 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*, 2008, pp. 27–30.

[8] 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)*, 2010, pp. 301–310.

[9] 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)*, 2007, pp. 34–43.

[10] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?" in *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM'08)*, 282008-oct.4 2008, pp. 337 –345.

[11] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10) - Volume 1*, 2010, pp. 45–54.

[12] 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)*, 2007, pp. 499–510.

[13] 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)*, 2008, pp. 461–470.

[14] J. Anvik, L. Hiew, and G. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse'05)*, 2005, pp. 35–39.

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

[16] 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)*, 2009, pp. 131–140.

[17] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM'10)*, 2010, pp. 1–10.

[18] 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)*, 2011, pp. 395–404.

[19] 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*, 2010, pp. 495–504. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806871

[20] 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," in *17th Working Conference on Reverse Engineering (WCRE'10)*, 2010, pp. 249–258.

[21] 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)*, 2007, pp. 364–374. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.74

[22] 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)*, IEEE. IEEE Computer Society, October 2010, pp. 249–258.

[23] 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)*. ACM, August 2009, pp. 135 – 143, amsterdam,The Netherland.

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

[25] 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.

[26] 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.