An Analysis of Committers toward Improving the Patch Review Process in OSS development

Shoji Fujita, Masao Ohira, Akinori Ihara and Ken-ichi Matsumoto
Graduate School of Information Science, Nara Institute of Science and Technology, JAPAN
{*shoji-f, masao, akinori-i, matumoto*}*@is.naist.jp*

***Abstract*—In recent years, individual and business users have made widespread use of Open Source Software (OSS) products. As OSS products have become important for the modern society, users expect OSS developers to fix existing bugs in the products as soon as possible. The goal of our study is to develop a better understanding of the patch review process in OSS development and to provide insight on how to elect an appropriate developer as a committer who will have a great impact on the quality of OSS products. We conducted a case study of the PostgreSQL project to look at an actual patch review process and how committers contribute to the patch review process. As a result of this case study, we found that there are significant differences between committers and non-committers in the number of patch reviews and edits. We also found committers tend to review and edit patches many times, but are not always speedy about reviewing and editing patches.**

***Keywords*-open source software (OSS); committer; patch review process**

I. Introduction

In recent years, not only individual but also business users have made widespread use of Open Source Software (OSS) products. As OSS products have become important for the modern society, users expect OSS developers to fix existing bugs in the products as soon as possible. In general, the modifications to fix existing bugs are carried out as follows;

1) A developer or a user finds a bug in an OSS product and then reports it to a mailing list (ML) or bug tracking system (BTS) managed by the OSS project which has been developing the OSS product.
2) The developers on the project read the bug report and then create a patch (source code diff) to correct the bug and submit the patch to the ML or BTS.
3) Other (mostly core) developers review the submitted patch to confirm the correctness of the patch.
4) If the core developers in the project verify the patch, it is finally "patched" (applied) to the product.

Since some OSS projects receive a large number of bug reports every day [2], improvement of the bug modification process is a growing concern for OSS projects and their users. To support the bug modification process in OSS projects, many studies have tried to paint a precise picture of the process[3]. For instance, regarding (1) above, [4][5] investigated how to write a good bug report which makes bug modification easier. Regarding (2), [6] revealed relationships between size and acceptance rate of patches submitted by OSS developers.

However, there are few studies on (3) and (4) above, that is, the review process of patches. Sometimes, submitted patches are neglected due to the limited number of developers in OSS project [7]. Even if they are reviewed and verified, the bug report itself often is not closed [1]. This means that a bug reporter cannot know if patches for a bug fix were actually applied to the product. So, it is also important in improving the bug modification process to make the patch review process more efficient and transparent.

In this paper, we analyze how submitted patches are reviewed and finally applied to OSS products. We focus especially on the differences in performance between committers and non-committers (normal developers). Here, a "committer" is defined as a developer who has write permission to a source code repository in a version control system such as CVS or Subversion. Committers can directly change source code in the repository by applying verified patches to existing source code. Most OSS projects have several or more committers to disperse workloads for modifying OSS products. Increasing the number of committers allows OSS project to skip patch reviews to some extent because a committer can directly modify source code in the repository. It also reduces the time from patch submission to patch commitment because workloads for patch reviews are more dispersed between committers.

Normally, a developer is selected as a committer, through nomination from other committers and/or core members, as a result of admirable contributions to the project. Depending on the state of the project, members of the project need to carefully choose a new committer because the role of committer can have a great impact on the progress of the project. Therefore, it has not been easy to elect an appropriate developer as a committer from a number of developers.

We conducted a case study of the PostgreSQL project to look at an actual patch review process and how committers contribute to the patch review process. The main contributions of our analysis in this paper are to develop a better understanding of the patch review process in OSS development and, based on our analysis, to provide an insight on how to select an appropriate developer as a committer. We

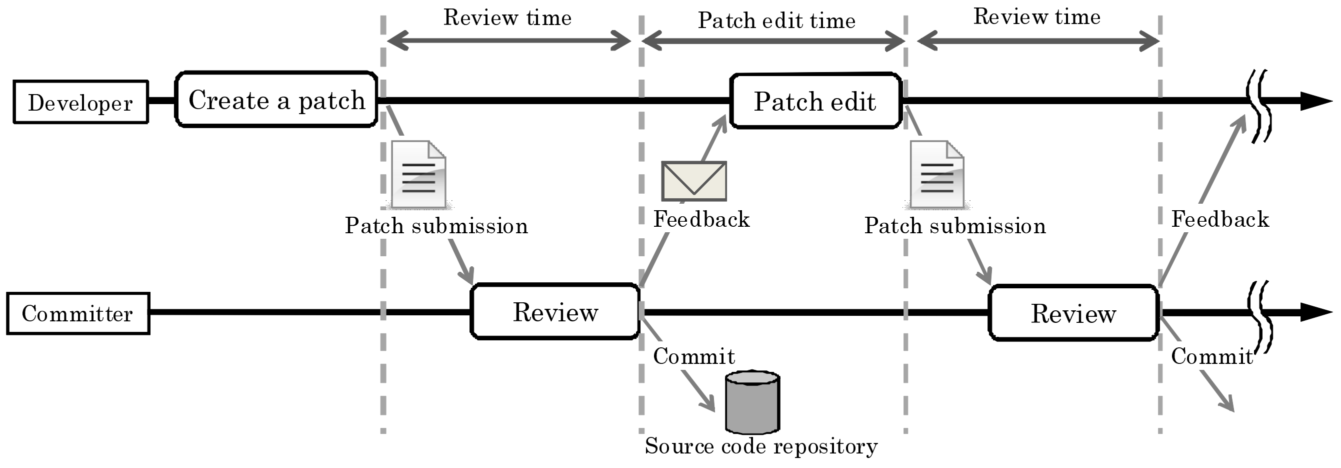


Figure 1: Patch Review Process

believe our findings will help OSS projects increase good committers, which would result in improvement of the patch review process.

In the following section, we describe the outline of the patch review process and define the terms used in this paper. Section III presents the research questions we have to answer in this paper. Section IV explains our analysis method. Section V shows the results for the case study of the PostgreSQL project. Section VI discusses the results. Section VII concludes our study and describes some ideas for future work.

II. Patch Review Process in OSS Development

This section describes an overview of the patch review process in OSS project and defines terms we use in this paper.

As described before, modification of OSS products is carried out by applying patches to existing source code. Figure 1 shows the patch review process where first a patch is submitted, then reviewed, accepted/rejected and finally applied to source code. In Figure 1, the horizontal line shows the time line in the process and the vertical line shows the actors who appear in the patch review process. In this paper we define "committer" as a developer who has commit permission, "non-committer" as a developer who does not have commit permission. We use "developer" when we do not distinguish committers and non-committers.

The patch review process basically consists of three steps:

1) One or more developers review the patch submitted by a developer to check if it is correct for removing bugs.
2) When the correctness of the patch is confirmed, a committer applies the patch to the corresponding source code and commits the patch to the source code repository.
3) When the patch is judged as not appropriate, a committer gives feedback to the developer who submitted the patch. The developer can then edit/rewrite and submit the patch again.

In Figure 1 the patch review process is defined as an iteration of two phases: Patch Review Time (PRT) and Patch Edit Time (PET). **PRT** is the time from patch submitted to patch confirmed as appropriate or the time from patch submitted to feedback given to a developer. **PET** is the time from feedback given to edited patch re-submitted.

III. Research Questions

In this paper we examined the following research questions to obtain better understanding of the patch review process in OSS development and the differences in performance between committers and non-committers.

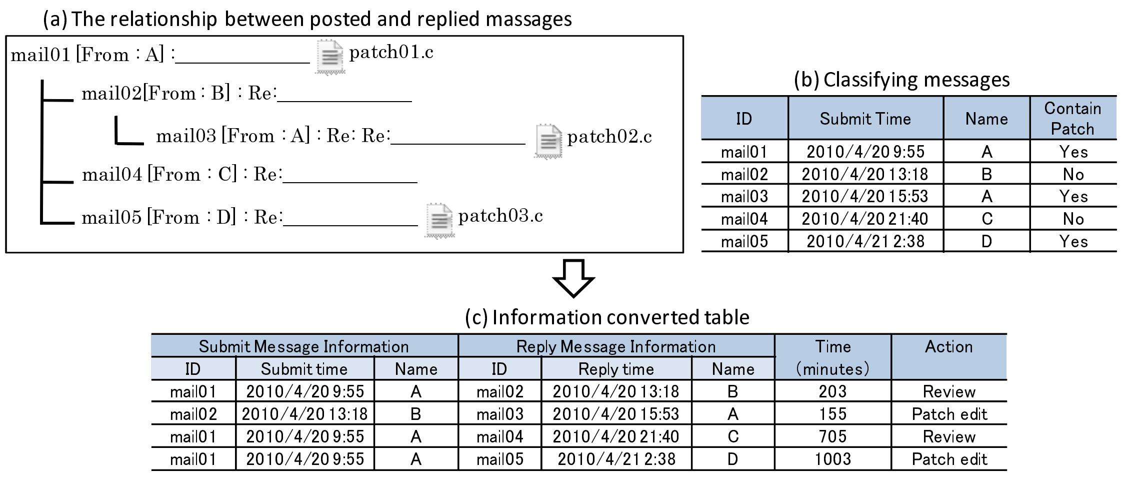
RQ1 Is there any difference of PRT and PET between committers and non-committers?

RQ2 Is there any difference in the number of patch reviews and edits between committers and non-committers?

RQ3 Is there any difference in performance between committers in the patch review process?

As described earlier, a developer is selected as a committer, through nomination by other committers and/or core members. To be nominated as a committer, s/he is expected to show active involvement in the project for a certain period. For instance, s/he has to actively join discussions in the project mailing list, submit a number of quality patches, and so forth.

So we may expect that the performance of committers in the patch review process is very different from that of non-committers. However, there are no explicit criteria to nominate a developer as a committer in most OSS projects. RQ1 and RQ2 were set up to understand the extent to which



ID	Submit Time	Name	Contain Patch
mail01	2010/4/20 9:55	A	Yes
mail02	2010/4/20 13:18	B	No
mail03	2010/4/20 15:53	A	Yes
mail04	2010/4/20 21:40	C	No
mail05	2010/4/21 2:38	D	Yes

Submit Message Information			Reply Message Information			Time (minutes)	Action
ID	Submit time	Name	ID	Reply time	Name		
mail01	2010/4/20 9:55	A	mail02	2010/4/20 13:18	B	203	Review
mail02	2010/4/20 13:18	B	mail03	2010/4/20 15:53	A	155	Patch edit
mail01	2010/4/20 9:55	A	mail04	2010/4/20 21:40	C	705	Review
mail01	2010/4/20 9:55	A	mail05	2010/4/21 2:38	D	1003	Patch edit

Figure 2: Procedure for Data Collection

the performance of committers and that of non-committers are different. And then we would like to find such the criteria in answering the questions.

Also, we would like to know about differences in performance among committers. We suppose there are a variety of ways in which committers contribute to the patch review process. For instance, one committer may submit a lot of quality patches and another committer may quickly review other developers ' patches and always give good feedback to them. By the same token, OSS projects may also have different needs for committers, because one project may face a lack of reviewers and another project may need more patch submissions. RQ3 was set up to understand the distribution of the differences of the performance between committers. If we can find such differences in performance between committers, we might be able to make a predictive model to suggest a candidate developer as a committer suitable to each project situation.

IV. Analysis

To answer the three research questions, in this paper, we analyze the patch review process, measuring PRT, PET, the number of patch edits and patch reviews. This section explains the data collection and our analysis method.

A. Data Collection

We collected the analysis data from the BTS or ML archives used in OSS development. Figure 2 shows the procedure for the data collection. Figure 2(a) shows the relationship between posted and reply messages in ML archive. If a message contains a patch, the filename of the patch is on the right side of the message. Figure 2(b) shows a table for classifying messages into messages with/without patches. Figure 2(c) shows a converted table using the information Figure 2(a) and (b).

When developer A posts the message “mail01” with the patch “patch01.c” and developer B replies to that message by “mail02”, “mail02” is defined as a feedback to the patch and defined that developer B reviewed the patch. In this case, PRT is calculated as the time from “mail01” posted to “mail02” replied. On the other hand, when developer D replies “mail05” with “patch03.c” to “mail01”, developer D is defined that s/he has edited the patch “patch01.c”. In this case, PET is calculated as the time from “mail01” posted to “mail05” replied. In this paper, we analyze posted and reply messages with patches, and analyze committers who can be identified in version control system, ML and BTS (i.e., committers who use the same account name in the systems).

B. Method

1) RQ1:PRT and PET: We collected the data of PRT and PET and calculate the median time of them for each developer. After that we used Mann-Whitney 's U-test to see if there is a significant difference of time between committers and non-committers. We only target the developers who have edited patches and/or reviewed more than 5 times.

2) RQ2 : Number of Patch Reviews and Edits: We collected the data of the number of patch reviews and edits and calculated the median number of them for each developer. Then we used Mann-Whitney 's U-test to see if there is a significant difference between committers and non-committers.

3) RQ3:Committer 's Patch Reviews and Edits: We define the criteria of PRT, PET, the number of patch reviews and edits, using all developers ' data. After that we classify committers by the criteria and analyze the differences in performance among committers.

V. Case Study

This section presents our case study of the PostgreSQL project to answer the research questions.

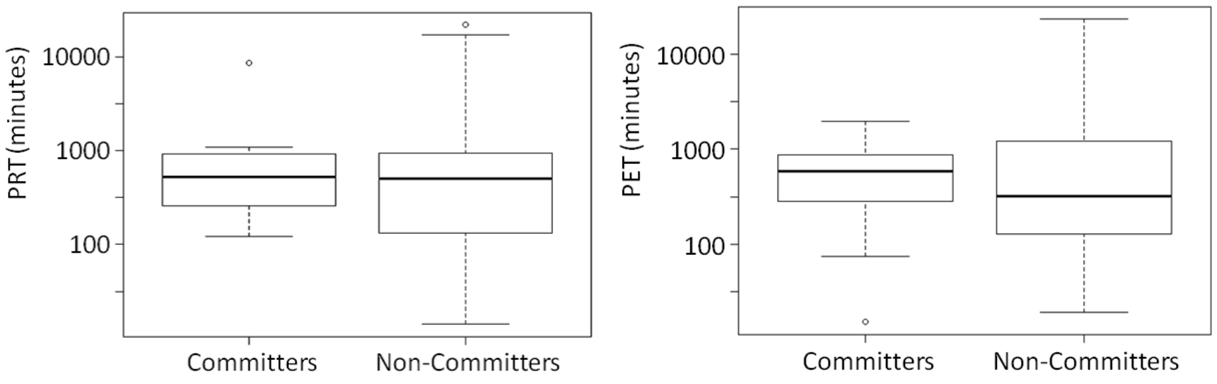


Figure 3: Patch Review Time (left) and Patch Edit Time (right)

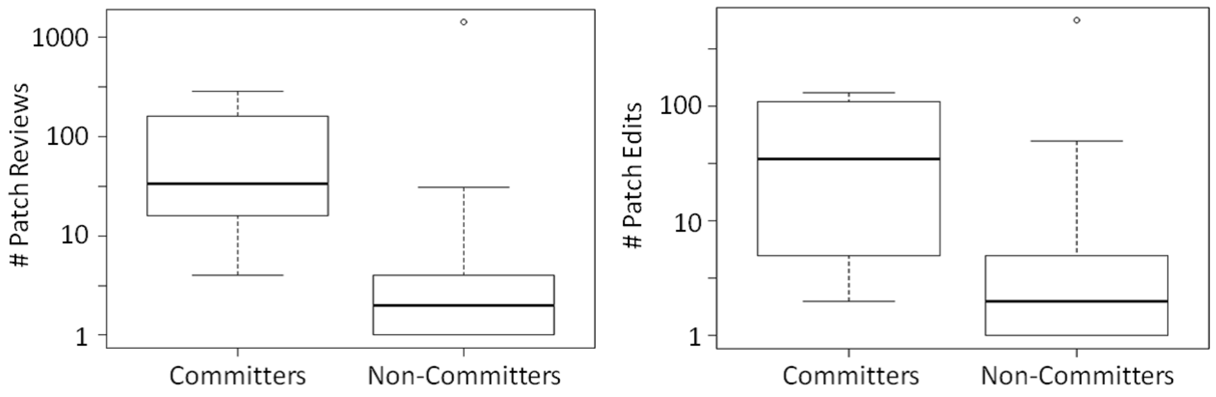


Figure 4: Number of Patch Reviews (left) and Number of Patch Edits (right)

A. *Target Data*

PostgreSQL is a widely used object-relational database management system. The analysis period was from January 2001 to December 2008 and we used two MLs “pgsql-hackers” and “pgsql-patches” for the target data. We could identify 14 committers who existed during the analysis period. We also could find 214 developers who reviewed patches and 146 developers who edited patches.

B. *Results*

1) RQ1：PRT and PET: Figure 3 shows distribution of PRT for committers and non-committers. In Figure 3 a box on the left shows distribution of PRT for committers and a box on the right is for non-committers. The vertical axis shows the median PRT and is converted into a logarithmic scale.There are 64 developers (14 committers and 50 non-committers) in Figure 3. Table I shows the median, average, standard deviation of PRT for the whole developers, non-committers, and committers. From the table, we cannot see a big difference of median PRT between committers and non-committers. The result of Mann-Whitney ’s U-test between committers and non-committers shows no significant difference (p=0.65 (>0.05)).

Figure 3 shows distribution of PET for committers and non-committers. In Figure 3 a box on the left shows distribution of PET for committers and a box on the right is for non-committers. The vertical axis shows the median PET and is converted into a logarithmic scale. There are 50 developers (14 committers and 36 non-committers) in Figure 3. Table II shows the median, average, standard deviation of patch edit time for whole developers, non-committers, and committers. From the table, the median of committer ’s PET seems to be shorter than non-committer ’s one. However, the result of Mann-Whitney ’s U-test between committers and non-committers shows there is no significant difference (p= 0.48 (>0.05)).

2) RQ2：Number of Patch Reviews and Edits: Figure 4 shows distribution of the number of patch reviews for committers and non-committers. In Figure 4 a box on the left shows distribution of the number of patch reviews for committers and a box on the right is for non-committers. The vertical axis shows the median number of patch reviews and is converted into a logarithmic scale. There are 214 developers (14 committers and 200 non-committers). Table III shows the median, average, standard deviation of the number of reviews for the whole developers, non-committers, and com-

Table I: Statistics of Patch Review Time

PRT	All developers	non-committers	committers
# Developers	64	50	14
Median	507.50 (2.71)	507.50 (2.71)	533.00 (2.72)
Average	1479.07 (3.17)	1579.25 (3.20)	1121.29 (3.05)
Standard dev	3572.63 (3.55)	3875.23 (3.59)	2132.97 (3.33)

Table II: Statistics of Patch Edit Time

PET	All developers	non-committers	committers
# Developers	50	36	14
Median	411.00 (2.61)	324.50 (2.51)	583.00 (2.77)
Average	1188.88 (3.07)	1394.47 (3.14)	660.21 (2.82)
Standard dev	3317.03 (3.52)	3877.61 (3.59)	493.46 (2.69)

Table III: Statistics of the number of Patch Reviews

# Patch Reviews	All developers	non-committers	committers
# Developers	214	200	14
Median	2.00 (0.30)	1.00 (0.00)	35.00 (1.54)
Average	15.55 (1.19)	10.56 (1.02)	86.93 (1.94)
Standard dev	100.93 (2.00)	99.65 (1.99)	91.79 (1.96)

Table IV: Statistics of the number of Patch Edits

# Patch Edits	All developers	non-committers	committers
# Developers	146	132	14
Median	2.00 (0.30)	1.00 (0.00)	34.50 (1.54)
Average	12.62 (1.10)	8.24 (0.92)	53.93 (1.73)
Standard dev	50.24 (1.70)	48.33 (1.68)	49.18 (1.69)

mitters. From the table, the median number of committer ’s reviews is much more than non-committer ’s. The result of Mann-Whitney ’s U-test between committers and non-committers shows a significant difference (p=0.00 (<0.05)).

Figure 4 also shows distribution of the number of patch edits for committers and non-committers. In Figure 4 a box on the left shows distribution of the number of patch edits for committers and a box on the right is for non-committers. The vertical axis shows the median number of patch edits and is converted into a logarithmic scale. There are 146 developers (14 committers and 132 non-committers). Table IV shows the median, average, standard deviation of the number of patch edits for the whole developers, non-committers, and committers. From the table, the median number of committer ’s patch edits is much more than non-committer ’s. The result of Mann-Whitney ’s U-test between committers and non-committers shows a significant difference (p=0.00 (<0.05)).

3) RQ3：Committer ’s Patch Reviews and Edits: Figure 5 shows the relationship between committer ’s PRT and PET. The horizontal axis shows PET and the vertical axis shows PRT. The horizontal line and vertical line in the figure show the median time of each PRT (508minutes) and PET (411minutes) as criteria. We classified committers using

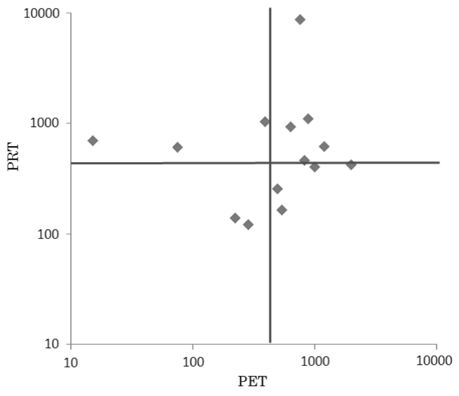


Figure 5: Committer Classification by the Patch Review Time and Patch Edit Time

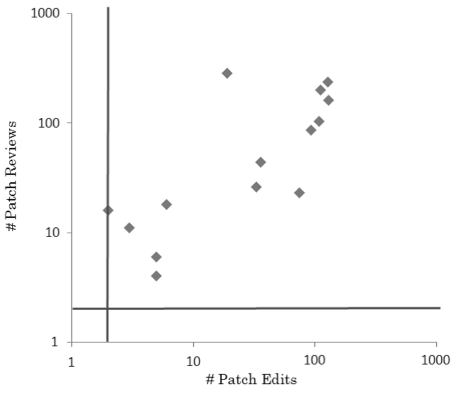


Figure 6: Committer Classification by the Number of Patch Reviews and Edits

criteria and the results show that there are 5 committers with longer PRT and PET, 4 committers with shorter PTR and longer PET, 3 committers with longer PRT and shorter PET, and 2 committers with both shorter PRT and PET. The results indicate that committers have a variety of PRT and PET.

Figure 6 shows the relationship between the number of patch reviews and edits by committers. The horizontal axis shows the number of patch edits and the vertical axis shows the number of reviews. The horizontal line and vertical line in the figure show the median number of reviews (2 times) and patch edits (2 times) as criteria.

From the results of classification by criteria, there are all 14 committers with more number of patch reviews and edits than the criteria.

VI. Discussions

In answering RQ1, we found that there is no big difference of PRT and PET between committers and non-committers. This could be because OSS development does not require a deadline and developers do not necessarily have to do a quick patch review and edit all the way along, even if they are committers who have a large effect on the productivity and quality of OSS products.

In answering RQ2 and RQ3 (in particular, the relationship between the number of reviews and edits by committers), we found that committers tend to have more patch reviews and edits than non-committers. The results suggest that committers in the PostgreSQL project win the admiration of their contributions in terms of the number of patch reviews and edits.

These results were not surprising to us, but a part of the results of RQ3 was unexpected. We found that the committers in the PostgreSQL project are not always speedy at reviewing and editing patches, and more than half number of the committers tend to take longer time for editing patches than criteria. This might happen because PostgreSQL is a database software product for which high reliability and performance are required. The PostgreSQL committers might have to be very careful to create quality patches.

For confirmation, we looked closely at the committer with the longest PRT in Figure 5 and found that he is a member of the PostgreSQL main contributors and the largest number of commits are done by him. He is likely to take a long time to take care of many patches. This could be the reason why his PRT is longer than other committers.

The findings imply that committers can contribute to OSS projects even if they have long PRT and PET. Not only the number of patch reviews and edits but also PRT/PET have to be evaluated to select a committer depending on the state of the project and/or product properties.

In this paper, we describe a case study of the PostgreSQL project, but we are aware of the weakness in the generality of our findings. The PostgreSQL project always has a restricted number of committers which is one of unique aspects of the project, but other OSS projects do not need to have such the restriction. Also, the PostgreSQL project uses a mailing list for patch submissions, but others often use a bug tracking system. These differences between PostgreSQL and other projects are likely to influence the results of our analysis. We have to investigate other projects in the future.

Due to the space limitation, in this paper we did not focus on the difference of the acceptance rate of patches between committers and non-committers. In order to choose an appropriate developer as a committer, the ability to create quality patches should be considered. We need to analyze this aspect of the patch quality in the future.

VII. Conclusions and Future Work

In this paper, to improve the patch review process in OSS projects, we analyzed differences in performance between committers and non-committers in the PostgreSQL project. From the case study of PostgreSQL, we could determine:

- There are no significant differences between committers and non-committers for PRT and PET.
- There are significant differences between committers and non-committers for the number of patch reviews and edits.
- Committers tend to review and edit patches many times, but are not always speedy at reviewing and editing patches.

In future work, we should consider patch quality such as the number of bugs included in a patch and readability of a patch. We should also count the number of patch reviews and edits per specific period of time to investigate contributions for each developer. In addition we would like to show the criteria for selecting committers who will keep contributing to the patch review process in the future.

Acknowledgment

We appreciate the anonymous reviewers giving insightful comments and helpful suggestions.

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] A. Ihara, M. Ohira and K. Matumoto, *An Analysis Method for Improving A Bug Modification Process In Open Source Development*, In 10th International workshop on principles of software evolution (IWPSE2009), pp.135-143, 2009.

[2] B. Hailpern and P. Santhanam, *Software debugging, testing, and verification*, IBM Systems fournal, pp.4-12, 2002.

[3] C. Jensen and W. Scacchi, *Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study*, Proceedings of the 29th International conference on software engineering (ICSE2007), pp.364-374, 2007.

[4] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj and T. Zimmermann, *What makes a good bug report*, Proceedings of the 16th ACM SIGSOFT International symposium on foundations of software engineering (SIGSOFT2008/FSE-16), pp.308-318, 2008.

[5] P. Hooimeijer and W. Weimer, *Modeling bug report quality*, Proceedings of the 22nd IEEE/ACM International conference on automated software engineering (ASE2007), pp.34-43, 2007.

[6] P. Weisgerber, D. Neu and S. Diehl, *Small patches get in!*, Proceedings of the 5th International workshop on mining software repositories (MSR2008), pp.67-76, 2008.

[7] Y. Wang, D. Guo and H. Shi, *Measuring the evolution of open source software systems with their communities*, ACM SIGSOFT Software Engineering Notes, Vol.32, No.6, p.7, 2007.