Peer Code Review in Open Source Communities Using ReviewBoard

Amiangshu Bosu
Department of Computer Science
The University of Alabama
Tuscaloosa, AL
USA
asbosu@ua.edu

Jeffrey C. Carver
Department of Computer Science
The University of Alabama
Tuscaloosa, AL
USA
carver@cs.ua.edu

Abstract

Peer code review is an effective method to reduce the number of defects and maintain source code integrity. Peer reviews in most of the Open Source Software (OSS) communities are conducted via mailing list, which are difficult to manage at times. Code review tools aim to ease the review process and keep track of the review requests. In this paper, we describe preliminary results of our study to evaluate code review process using a popular open source code review tool (ReviewBoard) in OSS communities. Some of our study findings are similar to the findings of previous studies on code reviews. In the projects under our study, we found that, most of the revisions are not submitted for peer review. More than 80% of the review requests are responded by two or less number of reviewers. Top committers of the projects are also top contributors of code reviews. Most of the review requests get prompt feedback within a day; however, some requests might wait for feedback for a long time. Most importantly, we have identified some interesting directions for future research.

Categories and Subject Descriptors K.6.3 [**Software Management**]: [Software development; Software Maintenance; Software Process]

General Terms Performance, Experimentation, Verification.

Keywords *Open source software, peer review, code review, mining repositories, review board*

1. Introduction

Software inspection (formal peer review) is considered to be an important and cost effective method of identifying defects [9]. The practice of using peer-reviews of code has been adopted as an important quality assurance mechanism in most of the mature and successful Open Source Software (OSS) communities [7, 12]. Besides detecting defects and maintaining the integrity of the code, peer-review helps the community spread knowledge, expertise, and development techniques among the review participants [16, 17].

Based upon when the review occurs, there are two types of peer-review processes used in OSS communities: 1) Pre-commit review: Code is committed to the repository only after peer(s) have reviewed it and are satisfied with it. This type of review process is also known as Review-Then-Commit (RTC). 2) Post-commit review: Code is inspected after it is submitted to the repository, usually a development branch. This type of review process is also known as Commit-Then-Review (CTR) [6, 14].

Most OSS communities conduct their peer-reviews via the mailing list. In this type of review, a code author generates a diff file or patch file (a type of file that can be generated by the source code versioning system, which represents changes made to the working copy of the code). Then the author sends the patch out via email using a special, designated keyword in email subject (e.g. Apache developers use '[PATCH]' as the keyword). The main drawback to mailing list-based code review is that management of the process is inefficient and prone to lost requests. Therefore, some OSS communities use tools specifically created for managing the code review process. Code review tools aim to manage the review requests and make the review task easier. Moreover, code review tools can maintain traceability link between review request and code revisions. ReviewBoard is one of the most popular open source code review tool used by OSS communities. This paper analyzes how OSS communities use ReviewBoard.

Section 2 briefly discusses the history and usage of ReviewBoard. Section 3 presents the research questions. Section 4 describes the research methodology. Section 5 analyzes the data relative to the research questions. Finally, Section 6 discusses the plan for future work and concludes the paper.

2. History and Workflow of ReviewBoard

Prior to 2007, the developers of VMWare used a time-consuming code-review process. They created htmldiffs by placing the old code and the new code side by side and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PLATEAU'12, October 21, 2012, Tucson, Arizona, USA.
Copyright © 2012 ACM 978-1-4503-1631-6/12/10…$15.00.

Pre-Print

highlighting the differences. Then, they emailed the htmldiffs to targeted reviewers along with an explanation of the change and the testing done for it. However, this process was difficult to maintain and resulted in a large number of lost requests. To ease the review process and to keep track of the review requests Christain Hammond and David Trowbridge released ReviewBoard in 2007 [8].

ReviewBoard is written in Python using the Django web framework, hosted on Google Code, and published under the Open Source MIT license [1]. The official project website provides project guidelines, documentation, developer blogs, and a list of companies and projects using the tool [2]. ReviewBoard is actively under development with regular releases. The latest stable version of ReviewBoard is 1.6.12, which was released on September 24, 2012.

The general workflow of a code review using ReviewBoard is shown in Figure 1. This workflow assumes pre-commit review mode. A review process starts after an author has made changes in his local source tree. He creates a review request using the diff file generated from comparing his changed files against the base code. In ReviewBoard, the author also must specify the absolute path in the repository used to generate the diff file. After the review request is published, the author has to wait for reviewers' feedback. Potential reviewers can see the request in the list of review requests. ReviewBoard notifies the code author when a reviewer has performed their review. If reviewers are satisfied with the code, the author submits the code to the trunk and closes the review request by marking it as "submitted". If the reviewer has asked for changes, the author makes the suggested changes and generates a new diff file. The author uploads the new diff file along with a description of the changes. This process of making requested changes and uploading diff repeats until the reviewer is satisfied.

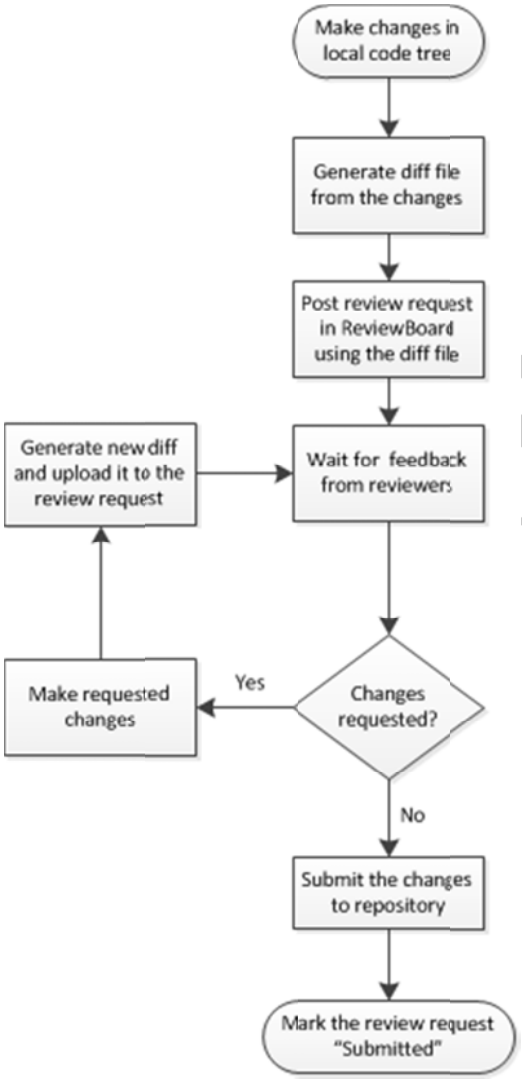

Figure 1: General workflow of code review in ReviewBoard

ReviewBoard can also be used in post-commit review mode. Projects using post-commit reviews typically use separate development and production branches. In the post-commit mode, a developer commits modified code to the development branch. Then, developers use RBTools (a companion tool of ReviewBoard) to create a review request based upon the code committed to the repository. A code change is merged to the production branch only after it has been approved by the reviewers.

3. Research Questions

Despite widespread adoption of peer-reviews in OSS communities, there has been little empirical research on the use of various peer-review techniques in OSS communities [14]. Most of the studies that do investigate the use of peer review in OSS focus primarily on mailing list-based peer-reviews [7, 13, 14]. In these studies, researchers collect data by mining mailing list archives using some assumptions (e.g. all requests will have specific keyword in the subject or will contain a diff file as an attachment). Therefore, there is not guarantee as to the completeness of the data [14]. We believe that data mined from ReviewBoard will be more accurate than data mined from mailing lists because all comments and interactions specific to a particular review request can be found in a single page. Moreover, use of ReviewBoard data provides traceability links between a review request and the corresponding code revision(s). Therefore, ReviewBoard data provides more insights into the review process. To our knowledge, no study has empirically evaluated the use of code review tools in the context of OSS code-reviews.

A number of major OSS projects (e.g. Apache, Asterisk, and KDE) have used ReviewBoard for a substantial period of time. We believe that mining ReviewBoard data will provide a rich dataset consisting different characteristics of the peer code review process in the OSS communities. The ultimate goal of our research is to understand the impacts of the code review process in OSS communities. For example, there are different characteristics associated with a code review request (e.g. first feedback time, number of comments received, review interval, rigor of the feedback, and acceptance). Those characteristics might be influenced by a number of factors (e.g. code length, module complexity, identity of the submitter, and number of active members in a community). We want to identify the type of relationship among the characteristics of the code review and the other factors. Again, the level of use and rigor of code review practices might also impact other characteristics of an OSS

e-Print

project, e.g. code quality, and number of defects found. Our goal is to identify the empirical evidence about those impacts. Once we better understand the current review process and the influencing factors, we will be better able to make suggestions to improve it.

This research is in its preliminary stage. The research questions in this paper are focused on descriptive statistics about the code-review process using ReviewBoard. The answers to these research questions will motivate additional research questions for the next step of our research. Some of the questions are motivated by results of previous studies about mailing list-based code reviews [7, 14]. We can compare the results of these common questions to determine whether there is any systematic difference between reviews conducted via the mailing list and reviews conducted with ReviewBoard. The remainder of this section describes and motivates each of the eight research questions addressed by this study.

RQ1. How many people are participating (i.e. submitting review request or reviewing code submitted by others) in the code-review process compared to the number of code committers in the project?

In an OSS project, only active developers can commit code directly into the main repository. Therefore, we expect that all code committers will participate in the ReviewBoard activities if the project requires all code to be reviewed before submission to the main repository. To answer this question, we collect two types of data: 1) whether all active developers participate in the ReviewBoard, and 2) whether there are people in addition to the code committers who participate in the ReviewBoard.

RQ2. Are all code revisions submitted for peer review?

Ideally, all code committed to the project's trunk should be peer-reviewed. However, there might be some exceptions to this rule. For example, a code author might not create review requests for trivial changes. Alternatively, a project member who contributes heavily to the project and who is aware of design and coding guidelines might not create review request for all of her changes. In this question, we determine what proportion of the code revisions are submitted for peer review in the ReviewBoard.

RQ3. Do the committers submitting the most revisions also contribute the most in the ReviewBoard?

If most code revisions are submitted for peer-review, then we would expect that the top code contributors would also be the top review request submitters. If a top committer is among the most active members, he might also be one of the top reviewers.

RQ4. What proportion of code revisions posted for reviews are not accepted for inclusion into the trunk?

In a pre-commit review, not all the changes submitted for review will eventually be committed to the repository. In this research question, we determine what percentage of the code for which a review is requested does not make it to the main repository of project.

RQ5. How many reviewers respond to a review request?

The Apache project RTC policy indicates that three core members of the project should review a code change prior to its inclusion in the main trunk. However, previous research indicated that this policy is not always followed [14]. Previous studies on the OSS review process found that on average 2 to 3 developers respond to a code review request [7, 13]. We revisit this question to determine how many reviewers typically respond to a review request in a ReviewBoard.

RQ6. How many comments do the reviewers and submitters make before code is approved for inclusion in the trunk?

After a review request is submitted, a code author might make several changes before the code is finally accepted. A reviewer may make comments requesting changes to the code. The code submitter may respond to those comments and upload new diff files incorporating those changes. When the reviewer(s) agree with the changes, the code is marked for shipping (i.e. marked for inclusion to the production branch). Again, a code review request may be reopened after it is marked as "Ship it" if a defect is found or if the code author identifies a better way of solving the problem. In each case, comments would be added to the review request. Previous study indicates that the average number of comments in a review request thread was between two and three [7]. We revisit this question to determine how many comments reviewers and submitters typically make about a review request in ReviewBoard before the change is accepted.

RQ7. How long does it take for a code author to receive the first feedback?

The review process adds a delay to committing the code to the repository. If a code author does not receive timely feedback about her code changes, it might slow down the project. Therefore, it is important for a code-review request to be completed quickly. In this research question, we explore how quickly a reviewer gets her first feedback from a review request.

RQ8. What is typical review interval using ReviewBoard?

Poter et al. defined "review interval" as the time from the start of a review process to its completion [11]. A review process starts when the code author prepares an artifact for review and ends when the code author has addressed all the concerns and changes requested by the reviewer(s). As preparing a diff file for a review request using the version control system does not take much time, we calculate the review interval as the time from the submission of the review request until the final comment is recorded in ReviewBoard. This question was also explored in a previous study [14].

4. Research Method

We used a two-stage research method. First, we created a list of candidate OSS projects that met two criteria: 1) the project has been actively using ReviewBoard for at least 1 year, and 2) the project's ReviewBoard data is publicly accessible. Second, to collect data, we developed a Java application to mine all the review requests posted to the ReviewBoards and insert the results into a MySQL database. Third, we conducted the mining of all the available review requests from the ReviewBoards during the last

Pre-Print

week of July 2012. Data analysis is still ongoing. This paper includes the results from two OSS projects: MusicBrainz Server [3] and Asterisk [4]. We selected these two projects because they were in active development and their ReviewBoards have a large number of review requests posted. Both of these projects mostly use pre-commit review process. If we obtain encouraging results from these projects, we plan to extend our analysis to other projects.

The MusicBrainz server project is an open content music database written in Perl. Currently, the MusicBrainz database contains information about 660,000 artists, 1 million releases, and 11 million recordings. Robert Kaye started this project in 2002. The current size of the codebase is around 90KLoc. The MusicBrainz project has been using ReviewBoard for the code review activities since April, 2009.

The Asterisk project is an open source platform for creating communication applications. It is written in C. Asterisk is currently the leading Open Source PBX software. Mark Spencer of Digium Inc. created the project in 1999. The first major release of the project was in 2004. The current size of Asterisk's codebase is around 840KLoc. The Asterisk project has been using ReviewBoard since October, 2008.

To collect statistics on code commits, we mined the code repositories of these two projects using the OSS tool CVSAnalY [15]. For better comparison of the trends, we selected the same time interval for both projects. The first full month of data available for the MusicBrainz ReviewBoard was May 2009. Therefore, we selected May 2009 as the starting point of our analysis interval. To ensure enough time for the completion of the review requests, we limited our analysis for the data until the end of April 2012. Therefore, in our analysis we used the data from May 2009 to April 2012 (36 months).

5. Analysis Results

This section presents the results of the data analysis related to the eight research questions based upon the data mined from the repositories of the two projects described in the previous section. Along with the answers to each question, we also pose new questions that will be evaluated in future studies.

5.1 How many people are participating in the code review process compared to the number of committers in the project?

Table 1 presents the number of committers and review participants during the three-year period included in our analysis. We found that 24 contributors committed code in the MusicBrainz server repository. During the same period, 23 contributors participated in the code review process. After crosschecking the two lists, we found that all the 23 contributors who participated in the code review process, also made code commits. The committer who did not participated in the code review process only made a small numbers of commits.

In the Asterisk project, 57 contributors committed code during the three-year period included in the analysis. During the same interval, 113 people participated in the Asterisk ReviewBoard. After crosschecking the two lists we found 46 contributors in common to both lists. We also noticed that, each of the top 15 code committers also participated in the Asterisk ReviewBoard.

Table 1. Number of committers compared to review participants

Project	Total # of Committers	Total # of review participants	Participants common in the both lists
MusicBrainz Server	24	23	23
Asterisk	57	113	46

The presence of people in the Asterisk ReviewBoard who do not have commit privileges can be explained by the Asterisk's ReviewBoard policy [5]. The Asterisk, ReviewBoard is also used for patch submission. Any interested contributor can submit a patch directly to the ReviewBoard to obtain feedback and acceptance of their code.

Based upon these observations, we can conclude that most of the core committers in the two projects also participate in the ReviewBoard activities. This situation might be ideal for projects with strong code review policies. It would be interesting to check whether this relationship holds true for other projects in our study and whether it holds for projects that use mailing list-based code reviews. These analyses will help determine whether ReviewBoard facilitates larger participation of the code authors in the review process.

5.2 Are all code revisions submitted for peer review?

During the three years period included in our analysis, 1150 code review requests were submitted to the MusicBrainz server ReviewBoard compared with 9105 revisions in the MusicBrainz official Git repository. During the same interval, 987 requests were posted in Asterisk ReviewBoard compared with 4595 source code revisions in Asterisk Subversion trunk. We know that some of the code submitted in a review request can be shipped more than one time, which results in multiple code revisions. However, we found that the proportion of such requests was relatively small (less than 15%). Again, some review requests are never accepted and those requests do not generate a new revision in the repository. Even though these two projects have strong code review policies that require all changes to be reviewed, we found that most of the changes were not submitted for review in these projects. It would be interesting to find out the proportion of reviewed code changes and unreviewed changes in other projects.

We found that code authors do not always submit their code changes for review. Therefore, a code author uses some criteria to decide whether his code needs a peer-review review. In the future, we want to analyze the factors that affect a code author's decision to submit a code change for peer review.

Preprint

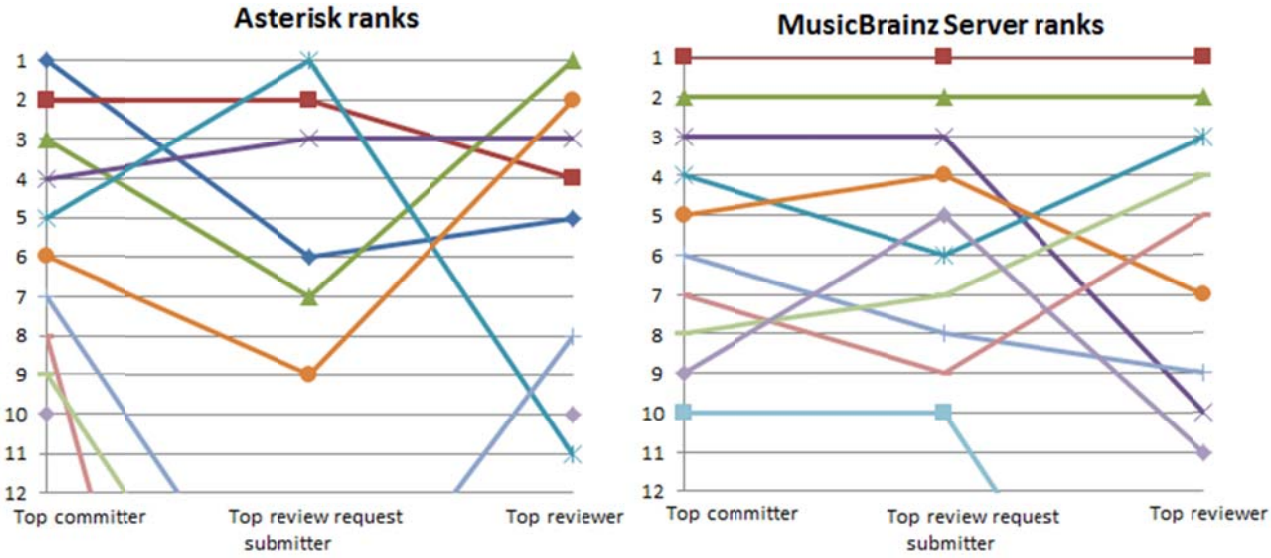

Figure 2. Comparing three different rankings of individual contributors in Asterisk and MusicBrainz server. Each series is created by connecting three different ranking of an individual in the three lists

5.3 Do the committers submitting most revisions also contribute the most in the ReviewBoard?

For each of the projects, we created three lists: 1) the top ten committers, 2) the top ten review request submitters, and 3) the top ten reviewers. Figure 2 compares the rankings of individual contributors in the three lists. These lists show that each of the top ten committers from the MusicBrainz server project are also in the list of the top ten review request submitters. Eight of the ten top committers also belong to the list of the top ten reviewers. In the Asterisk project, only six of the top ten committers belong to the list of the top ten review request submitters. Seven of the top ten committers also belong to the list of the top ten reviewers. Therefore, we conclude that contributors who make the most code contributions also tend to make the most contributions during the code-review process.

If all the code changes were submitted for review, the top committer rankings and top review request submitter rankings would be exactly same. The ranking of the top contributors across the three lists are more similar in MusicBrainz server project than in the Asterisk project. We believe this result may be due to higher proportion of code changes submitted for reviews in MusicBrainz server (21%) than in the Asterisk project (13%). We plan to investigate this belief further.

Again, for the top individual contributors we can generate their contribution graphs across time interval (e.g. number of code commits/review requests during three months interval). This data will help us answer some interesting questions. For example:

- Is the ratio of code commits and review requests similar for a contributor across different time intervals?
- As a code contributor becomes more experienced, does she tend to submit a smaller portion of her code changes for review?
- Do members review more other contributor's code as they become experienced?

5.4 What proportion of code revisions posted for reviews are not accepted for inclusion into the trunk?

In the MusicBrainz server project only 1 out of the 1150 requests were not accepted for submission during the three-year period. Conversely, in the Asterisk project, 74 out the 987 reviews (7.5%) were not accepted for submission. We believe that the higher percentage of rejected code in the Asterisk project is due to Asterisk's ReviewBoard usage policy. As mentioned earlier, anyone can submit a patch to the Asterisk ReviewBoard. In the MusicBrainz server, only trusted project members who have code commit privilege can submit review requests. It is important to note that this type of analysis is not possible for data mined only from mailing lists.

5.5 How many reviewers respond to a review request?

Figure 3 presents the cumulative distribution of the number of people responding to a review request. In the MusicBrainz server project, 7.5% of the accepted review requests were not reviewed by any reviewer. Approximately 65% of the review requests were reviewed by only one reviewer. More than 93% of the review requests involve only one or two reviewers apart from the code author. In the Asterisk project, around 5% of the accepted review requests were not reviewed by any reviewer. Around 55% of the review requests in the Asterisk project were reviewed by only one reviewer. Only one or two reviewers reviewed more than 82% of the review requests in the Asterisk project. These finding are very similar to the findings of Rigby et al.'s study of the Apache server project. They found 50% of the requests were reviewed by only one individual and 80% were reviewed were reviewed by two [14].

The average number of people responding to a review request in the MusicBrainz server project is 1.28 people. In

Pre-Print

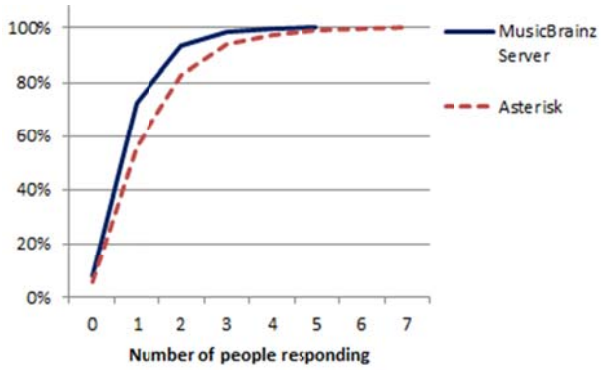

Figure 3. Cumulative distribution of number of people responding to a review request

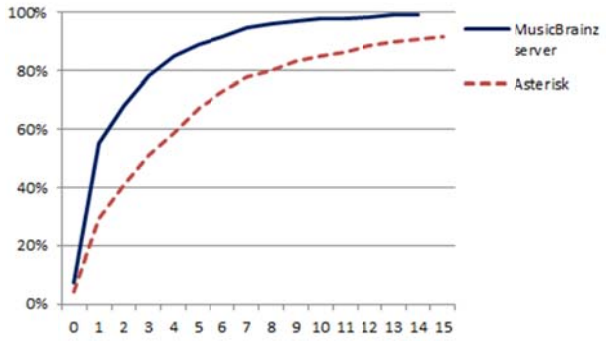

Figure 4. Cumulative distribution of number of comments posted in a review request

the Asterisk project, the average number of respondents to a review request is 1.49. This result is similar to the findings of Ashundi et al. The average number of respondents in Apache httpd, GCC, GDB, Mplayer, and Gaim were 1.81, 1.26, 1.31,1.77, and 0.94 respectively [7].

We believe the size of the core development group in a community is positively correlated with the average number of people responding to review request. In the Linux kernel, which has a larger core group, the average number of responses to a review request (2.35) was higher than the number found in our study [10]. We also find support for this belief in the results of our study. The number of committers in the Asterisk project is higher than the number of committers in the MusicBrainz server project. We found that, on average, more reviewers participated in the review requests in the Asterisk project than in the MusicBrainz server project. To substantiate this claim we need to do a similar analysis on a larger number of projects.

Other factors might also influence number of reviewers that respond to a review request. For example: popularity of the code author, popularity of the module, and complexity of the code. It would be interesting research question to analyze whether these factors actually affect the number of reviewers that respond to a review request.

5.6 How many comments do the reviewers and submitters make before code is approved for inclusion in the trunk?

Figure 4 shows the cumulative distribution of the number of comments made on a review request. The number of review requests that did not receive any comments is around 7% in the MusicBrainz server project and around 4% in the Asterisk project. The average number of comments per review request is 2.5 in the MusicBrainz server project and 6.1 in the Asterisk project. In the Asterisk project around 60% of the review requests generate three or more comments. While in the MusicBrainz server project, only 30% of the review requests generate three or more comments.

About 15% of the review requests in Asterisk generate ten or more comments. While in the MusicBrainz server project, only 2.5% review requests generate ten or more comments. One reason for the higher number of comments in the Asterisk project could be the presence of a larger number of developers in the community. Of course, there could also be other reasons. A detailed analysis of the characteristics of the review requests that generate the largest number of comments would be interesting research topic.

In this phase of our study, we only counted the number of comments. We did not look into the length or contents of the comments. We believe it worth taking a detailed look at the contents of the comments. In the future, we want to analyze whether most of the comments were short (e.g. 'Looks good') or whether they contain constructive suggestions to improve the code changes.

5.7 How long does it take for a code author to receive the first feedback?

Figure 5 shows the cumulative logarithmic distribution of time interval between the submission of a comment and the first feedback in the two projects. The average time passed before first feedback on a review request is 3.61 days in the MusicBrainz server project and 5.04 days in the Asterisk project. However, the average time may not be a good indicator of how promptly the first feedback is received. A few requests receive their first feedback only after a long period of time, which makes the average value higher. The median of the first feedback may be a better measure. The median is 1.2 days for the MusicBrainz server project and 0.2 days (4.8 hours) for the Asterisk project.

In the MusicBrainz server project more than 20% of the review requests receive their first feedback within 2.4 hours (0.1 day), while in the Asterisk project more than 40% review requests receive first feedback within 2.4 hours. More than 71% review requests get first feedback within the first day in the Asterisk project. While only 47% review requests receive first feedback within the first day. For both of the projects, almost 90% of the review requests receive first feedback within first week. By comparing the Asterisk project with the MusicBrainz server project, we can observe that Asterisk project members are more prompt in providing the first feedback to the code review requests. However, for some of the review requests Asterisk members take a very long time. This type of delayed response happens less frequently in the MusicBrainz server project. We conclude here that in both projects most of the review requests receive prompt feedback, with only a few facing

Pre-Print

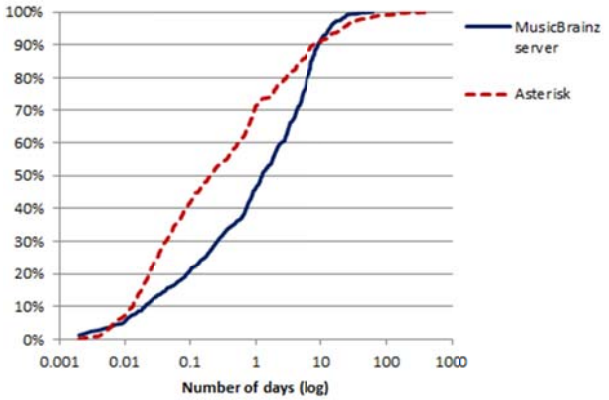

Figure 5. Cumulative logarithmic distribution of time for first feedback on a review request

long delays. Again, the larger core group of Asterisk may explain the lower median feedback time for review requests. We need to analyze similar data from more projects to validate this observation.

Again, other factors (e.g. popularity of the code author, popularity of the module, and complexity of the code) may influence the first feedback time of a review request. It would be interesting research question to check if the those factors indeed affect the first feedback time of review requests.

5.8 What is typical review interval using ReviewBoard?

Figure 6 shows the cumulative logarithmic distribution of review interval of the two projects. The average review interval in the MusicBrainz server project is 9.42 days and in the Asterisk project is 24.7 days. Again, the median should be a better indicator of the distribution of the review intervals in these projects. The median review interval in the MusicBrainz server project is 4.3 days and in the Asterisk project is 3.7 days.

For the MusicBrainz server project approximately 25% review requests are completed within the first day, while in the Asterisk project around 32% review requests are completed within the first day. Within ten days, approximately 75% review requests are completed in MusicBrainz server project and approximately 70% review requests are completed in Asterisk project. The review interval found in these projects is higher than what was found in the study of Apache server [14]. It would be an interesting question to analyze the difference between the characteristics of the short-lived and long-lived review requests. This information can help to predict factors that can help expedite the review process.

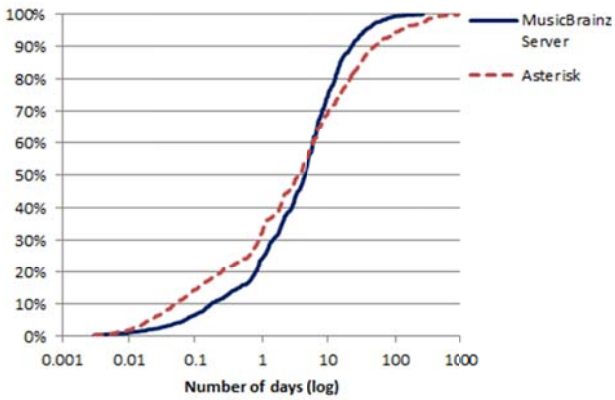

Figure 6. Cumulative logarithmic distribution of review interval

6. Future work and Conclusion

In this paper, we presented the preliminary results of our analysis of the use of ReviewBoard for managing the peer-review process in OSS projects. Three out of our eight research questions were motivated from previous studies. For two of those questions, we have found result similar to previous studies. We believe the preliminary results to be encouraging and worth more research. We plan to expand this analysis by including data from more projects to get a better idea of the review process in OSS communities using ReviewBoard.

During our analysis, we identified the following interesting research questions that should be explored in the future.

1. Not all the code revisions are submitted for review. Therefore, what are the differences in the characteristics of the revisions which are reviewed and the revisions which are not? How do code committers decide which code to submit for review?
2. The top committers are also among the top request submitters and top reviewers. Is this result applicable to other projects with different sizes or review processes?
3. What are the main differences between the characteristics of the review requests that are accepted and those that are not?
4. What are the factors that affect the number of reviews in a review request?
5. What are the factors that contribute to a larger number of comments in a review requests?
6. What proportions of review requests receive constructive suggestions to improve the code changes and what are the factors that affect those comments?
7. Which factors affect the first feedback time and review interval?

We believe data mined from projects using ReviewBoard will be helpful in answering these research questions.

References

[1] ReviewBoard on Google Code. http://code.google.com/p/reviewboard/ Accessed August 05, 2012.

[2] ReviewBoard official website. http://www.reviewboard.org Accessed August, 05, 2012.

[3] MusicBrainz official website. http://musicbrainz.org/ Accessed August 05, 2012.

[4] Asterisk official website. http://www.asterisk.org/ Accessed August 5, 2012.

Pre-Print

[5] Asterisk code acceptance policy. https://wiki.asterisk.org/wiki/display/AST/Reviewboard+Usage Accessed August 5, 2012.

[6] Code review glossary. http://www.reviewboard.org/docs/manual/dev/glossary/ Accessed August 5, 2012.

[7] J. Asundi and R. Jayant. Patch review processes in open source software development communities: A comparative case study." In *40th Annual Hawaii International Conference on System Sciences (HICSS 2007).*

[8] Christain Hammmond, "ReviewBoard release announcement,"

[9] M. Fagan. Reviews and inspections. *Software Pioneers--Contributions to Software Engineering* 562-573. 2002.

[10] G. Lee and R. Cole. From a firm-based to a community-based model of knowledge creation: The case of the linux kernel development. *Organization Science 14*(6): 633-649. 2003.

[11] A. Porter, H. Siy, A. Mockus and L. Votta. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering and Methodology (TOSEM) 7*(1): 41-79. 1998.

[12] P. C. Rigby and M. A. Storey. Understanding broadcast based peer review on open source software projects." In *Proceeding of the 33rd International Conference on Software Engineering.*

[13] P. C. Rigby and D. M. German. (2006, A preliminary examination of code review processes in open source projects. Technical Report DCS-305-IR, University of Victoria,

[14] P. C. Rigby, D. M. German and M. Storey. Open source software peer review practices: A case study of the apache server." In *Proceedings of the 30th International Conference on Software Engineering.* Leipzig, Germany Available: http://doi.acm.org/10.1145/1368088.1368162

[15] G. Robles, S. Koch and J. Gonzalez-Barahona. (2004, Remote analysis and measurement by means of the CVSAnalY tool. Working Paper, Informatics Department, Universidad Rey Juan Carlos,(June).[Available at: http://opensource. mit. edu/papers/robles-kochbarahona_cvsanaly. pdf.],

[16] S. Turner, M. A. P\'erez-Qui\nones, S. Edwards and J. Chase. Peer review in CS2: Conceptual learning." In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education.*

[17] K. Wiegers and B. Addison-Wesley. Peer reviews in software: A practical guide. *Recherche 67*02. 2001.

Pre-Print