The Impact of A Reviewer's Low Level of Agreement in a Code Review Process

Name of Author[1] and Name of Author[2]

[1] Name and Address of your Institute name@email.address
[2] Name and Address of your Institute name@email.address

Abstract. Code review systems are commonly used in software development projects to ease the review processes. In these systems, many patches are submitted in a daily basis for improving quality of source codes. To verify the quality of patches, voting is the common process used by reviewers and committers to make the final decision for a patch; however, there still exists a major problem in this process, that is reviewers' vote is not always simply reach a broad agreement. For example, in our previous study, we found an approximate of 59.5% of reviews were conflicted, implying that an individual reviewer's final decision for a patch usually differs from that of the majority of the other reviewers. In this study, we further investigate for the reasons to explain why such situations are often occurred, and provide suggestions for better handling these problems. Our detailed analyses on the Qt and OpenStack project datasets allow us to suggest that a patch owner should select more appropriate reviewers who often agree with others' decisions when he needs to invite reviewers for his patches. This is an important finding of future of code reviewing processes, where voting is the main criteria for final decision making.

1 INTRODUCTION

Source code review is a process in which reviewers and committers verify patches and decide whether or not a patch should be integrated into a version control system. In particular, Open Source Software (OSS) projects conduct a review to release higher quality and readability source codes [1]. Endorsed by McIntosh et al. [2], a code review with sufficient discussions is one of the most useful practices contributed for a more effective bug detection process. In addition, a code review has proofed to be very helpful in providing important feedbacks to developers for their future developments [8].

Nowadays, we have various dedicated tools for managing the code review process. For example, Gerrit [1] and ReviewBoard [2] are commonly used by OSS practitioners to improve the quality of their source codes. Technically, conducting a code review process using these tools is done through patch submissions and reviews, and voting is known to be one of the most commonly used practices

[1] Gerrit: https://code.google.com/p/gerrit/
[2] ReviewBoard: https://www.reviewboard.org/

to decide whether or not a patch should be integrated into a version control system (i.e., to be accepted). In general situation, patches with higher quality are more likely reached their final conclusion to be accepted faster than others; however, we seem to be unable to expect all the submitted patches to be high quality. In such cases, voting results can be varied and thus adding more difficulty to decide the conclusion for those patches. This is because, in particular, the final decision cannot be made unless reviewers and committers have reach their consensus.

Reported in our previous study [3], the consensus is not usually reached under voting system in practice. That is, we found an approximate of 59.5% of the reviewers' decisions were in disagreement with others, who made decisions prior to them. As a continuation of our previous study, this study further investigates how does a reviewer disagree with the other reviewers' decision, and what is the impact of a reviewer with a low level of agreement. Hence, we conduct a case study using Qt and OpenStack project datasets to address the following research questions:

RQ1: How often does a reviewer disagree with a review conclusion?
Results: A more experienced reviewer is likely to have a higher level of agreement that a less experienced reviewer.

RQ2: What is the impact of a reviewer with a low level of agreement in a code review?
Results: A review with a higher level of agreement is more likely taken shorter reviewing time and spent shorter discussing length.

This paper is arranged as follows. Section 2 describes the background to this paper, related work and motivating example. Section 3 provides the design of our two research questions and datasets. Section 4 presents the results with respect to our two research questions. Section 5 discusses a qualitative analysis of our research questions, and addresses the threats to validity. Finally, Section 6 concludes this paper and describes our future work.

2 BACKGROUND

2.1 Modern Code Review (MCR)

In recent years, MCR becomes popular and widely used in both proprietary software (e.g., Google,Cisco, Microsoft) and open source software (e.g., Android, Qt, LibreOffice) [4]. The use of these reviewing tools has made the review process more traceable, which has created opportunities for empirical software engineering researchers to analyze this process [5, 6, 2, 7, 8]. For example, Hamasaki et al. [5, 9, 10] collected rich code review datasets from a collection of open source projects using the Gerrit and ReviewBoard code review tools.

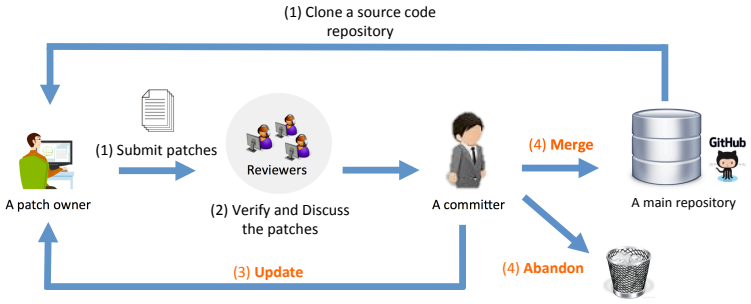

Fig. 1. An overview of Modern Code Review Process

2.2 Patches-related activities in MCR

Jiang et al. noted that a submitted patch is not always merged into a version control system [11]. Examples of abandoned patches include patches with unfixed bugs, irrelevant comments, or duplicate patches [12]. To verify the patches adequately, McIntosh et al. [2] suggested reviewers to discuss the patches carefully and try to reach a consensus in their discuss. In this way, final conclusion of a patch can be simply made based on the reviewers' discussion.

Figure 1 provides an overview of a code review process such as using a code review tool. In particular [8], the code review process after patch submission acts as follows:

(1) A patch owner clones a source code repository from a web-based version control system service such as GitHub to their local computers. Then, the patch owner creates patches to fix a bug or to enhance. After that, he submits the patches to a web-based code review tool.

(2) The patch owner requests reviewers to verify the submitted patches. The reviewers verify the changes. Then, the reviewers will post an approving positive vote or a disapproving negative vote, and sometimes also post comments.

(3) If the patch owner needs to revise his patches, he will update his patches.

(4) If the patches are approved by reviewers and a committer, the committer decides to ***Merge*** the patches into the main repository. On the other hand, if the patches are not approved by reviewers and a committer, the committer decides to ***Abandon*** the patches. In this paper, Merge and Abandon are referred to as a final review conclusion.

Nowadays, many researchers proposed approaches to improve a code review process. Weißgerber et al. [13] found that a smaller patch is more likely to be accepted. And, Tao et al. [12] investigated reasons why a patch was rejected through their quantitative and qualitative analysis. Furthermore, to verify a

Table 1. The voting patterns made in the final review conclusion in Qt.

	N = 0	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7
P = 0	56, 4268	25, 2090	16, 505	3, 93	2, 23	0, 2	0, 0	0, 0
P = 1	40425, 968	1475, 571	135, 122	19, 28	3, 6	0, 1	0, 0	1, 0
P = 2	11748, 247	920, 135	81, 38	20, 12	5, 0	1, 3	0, 0	0, 0
P = 3	2686, 51	316, 29	54, 6	8, 1	1, 0	0, 0	0, 0	0, 0
P = 4	587, 10	79, 8	17, 1	6, 1	1, 0	1, 0	0, 0	1, 0
P = 5	119, 2	36, 1	7, 1	1, 0	0, 0	1, 0	0, 0	0, 0
P = 6	17, 0	3, 0	4, 0	0, 0	1, 0	0, 0	0, 0	0, 0
P = 7	6, 0	2, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0

Table 2. The voting patterns made in the final review conclusion in OpenStack.

	N = 0	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7
P = 0	141, 20	4243, 3155	1193, 1214	351, 344	133, 116	47, 47	20, 9	2, 6
P = 1	11964, 1047	3497, 1444	1185, 752	436, 298	157, 122	67, 37	27, 16	8, 10
P = 2	6269, 264	**2254**, **467**	865, 302	324, 134	126, 64	54, 25	15, 14	9, 5
P = 3	3069, 90	1317, 211	514, 124	251, 68	85, 31	33, 25	15, 9	11, 2
P = 4	1519, 42	690, 106	303, 55	134, 39	51, 25	24, 7	11, 2	1, 3
P = 5	687, 30	357, 50	146, 37	86, 20	45, 13	13, 5	10, 2	1, 2
P = 6	370, 13	193, 36	91, 25	52, 13	15, 5	17, 5	8, 2	0, 2
P = 7	182, 10	116, 29	60, 17	36, 7	23, 5	4, 3	3, 3	3, 2

patch adequately, Thongtanunam et al. [8] and Xia [14] proposed approaches to identify a reviewer who should verify the patch.

2.3 Motivating Example: The Code Review Collaboration Among Reviewers

Based on practical observations explained in the following of this section, we are motivated to seek for a better understanding of the discussion disagreement among reviewers.

Table 1 and Table 2 show the number of patches of each voting pattern in the two projects when a committer makes the final review conclusion in Qt and OpenStack project data. Each cell has #Merged patches (A left value) and #Abandoned patches (A right value). These two tables show that the patches with less than seven positive votes or negative votes, then 91% patches in Qt and 91% patches in OpenStack are covered in whole patches. 15% patches in Qt and 31% patches in OpenStack of all patches do not follow the majority rule that selects alternatives which have a majority. For example, in Table 2, when #Positive is 2 (P=2) and #Negative is 1 (N=1), 2,254 patches are merged. However, even though #Positive is more than #Negative in this case, 467 patches are abandoned. It indicates that the negative votes might be disagreed with the final review conclusion ("Merge"). Therefore, we think that not all of reviewers post a vote agreed with the final review conclusion in a code review discussion. Therefore, in this paper, we study the impact of a reviewer with a low level of agreement in a code review collaboration.

3 CASE STUDY DESIGN

This study considers two research questions to understand the impact of a reviewer who disagrees with a review conclusion of a patch in a code review collaboration. The following in this section we provide detailed explanations of the case study designed to answer these research questions, and address our experimental Dataset.

3.1 Research Questions

RQ1: How often does a reviewer disagree with a review conclusion?

Motivation. Rigby et al.[15] concerned that code reviews are expensive because they require reviewers to read, understand, and assess a code change. Thongtanunam et al. [8, 14, 16] said that to effectively assess a code change, a patch owner should find an appropriate reviewer who has a deep understanding of the related source code to well examine code changes and find defects. In other words, the appropriate reviewer is more likely to assess the patches adequately. However, this reviewer might not always agree with a review conclusion.
Approach. To answer the first research question, we analyze a difference of a reviewer's agreement according to reviewer's experience (the number of votes in the past). A committer confirms the reviewers' votes to decide a final review conclusion (Merge or Abandon). To analyze the reviews's decision results, we scan comments for the known patterns of automatically-generated voting comments in Gerrit as shown in Table 3. Table 3 shows five level validating scores ("+2", "+1", "0", "-1", "-2") in Gerrit. A reviewer can vote "+1", "0" and "-1" score. On the other hand, a committer can vote "+2", "+1", "0", "-1" and "-2" score [17]. In our study, we use these vote scores "+2", "+1", "0", "-1" and "-2" to analyze assessments by reviewers and committers .

Next, to identify the frequency of votes disagreed with review conclusions, we track the voting history of each reviewer. We count the number of times that a reviewer disagreed with the review conclusion of a patch in the past. We need

Table 3. The patterns of automatically-generated voting comments in the Gerrit.

Role	Score	Automatically-generated voting comments
Committer	+2	"Looks good to me, approved"
		"Looks good to me"
Reviewer	+1	"Looks good to me, but someone else must approve"
		"Works for me"
		"Code-Review+1"
		"Workflow+1"
		"Verified"
	0	"No score"
	-1	"I would prefer that you didn't submit this"
		"I would prefer that you didn't merge this"
		"Code-Review-1"
		"Workflow-1"
		"Doesn't seem to work"
Committer	-2	"Fails"
		"Do not merge"
		"Do not submit"

Table 4. The definition of agreement and disagreement patterns

		Review conclusion		
		Merge	Abandon	Update
Rreviewer's vote	Positive (+2 or +1)	`agree`	`disagree`	`disagree`
	Negative (0, -1 or -2)	`disagree`	`agree`	`agree`

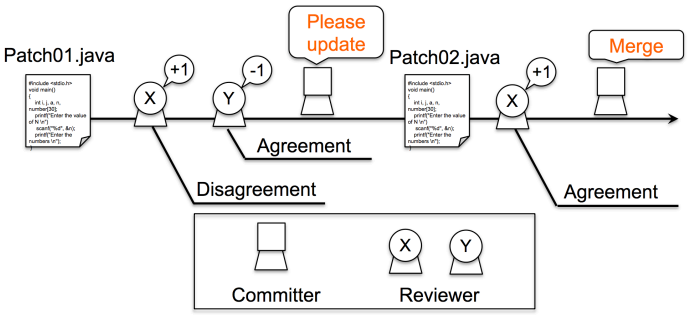

Fig. 2. An example of a voting process.

to be careful when calculating the frequency, because some patches have often been updated. In the case that a patch is updated twice, a reviewer will have two chances to post a vote. In this case, we count the votes twice. Figure 2 shows an example of a voting process. A reviewer X reviews two patches, namely Patch01.java and Patch02.java. As shown in the figure, his or her first positive vote is disagreed with the first review conclusion, which was decided to be "Updated", because a positive vote implies that this patch does not have any problem. After the patch is updated, his or her second vote is still positive. Finally, the vote of the reviewer X was agreed with the final conclusion, which is decided to be "Merged". In this case, the rate of agreed votes for the reviewer X is $\frac{1}{2}$=0.5 (Reviewer X's level of agreement = 50%). Also, a reviewer Y posts a negative vote for only Patch01.java. The rate of agreed vote for the reviewer Y is $\frac{1}{1}$=1.0 (Reviewer Y's level of agreement = 100%). A level of agreement has the range between 0.0 and 1.0. In summary, Table 4 describes the definition of the agreement and disagreement votes. If a reviewer posts a positive vote (+2 or +1) for a patch and a committer decides to merge this patch, the vote is an agreed vote. On the other hand, if the committer decides to abandon this patch or a patch owner updates this patch, the vote is a disagreed vote.

RQ2: What is the impact of a reviewer with a low level of agreement in a code review?

Motivation. In a code review, when a discussion does not always reach a consensus among reviewers and committers, it may take much longer time to completely finish the code review. In addition, it may be not simply to identify which vote that a committer should believe. In this second research question,

Table 5. Summary of the studied datasets

	Qt	OpenStack
Original datasets	70,705	92,984
At least 1 vote and without only bot test's vote	61,076	61,642
Without 10% sets	55,523	56,038

we investigate the impact of code review collaboration with a reviewer who often disagree with a review conclusion using the reviewer's level of agreement measured in the RQ1.

Approach. In this research question, we focus on a minimum level of agreement of reviewers calculated in each individual review. According to Figure 2, a minimum level of agreement of this example is the level of agreement of reviewer X's, which is 0.5. A minimum level of agreement has the range between 0.0 and 1.0. When the value of a minimum level of agreement is low, it can be implied that the review has a reviewer who often disagrees with a review conclusion. In our opinion, lower level of minimum agreement has a negative effect on the code review process. To further analyze the negative effect, we define two technical terms as follows:

Reviewing Time: The time in days from the first patch submission to the final review conclusion. We hypothesize that a reviewer with a lower level of agreement may take much longer time to reach a consensus in the discussion. To reduce the chance of a release postponement, *Reviewing Time* should be shorter.

Discussing Length: The number of comments which reviewers post into a reviewing board. We hypothesize that a reviewer with a lower level of agreement may lead to make a disagreement, so that such a code review needs much longer discussion to reach a consensus among reviewers.

3.2 Experimental Dataset

We conduct a case study on two large and successful OSS projects namely Qt [3] and OpenStack [4]. These two projects are commonly in the literature on OSS studies, such as in [2, 3, 5, 8, 9, 18], mainly because they have a large amount of reviewing activity using a code review tool.

Originally, Figure 5 shows that Qt and OpenStack have 70,705 and 92,984 review reports, respectively. However, we concern that reports consisting no vote or only bot test's vote are unuseful because we focus on a human reviewer. Therefore, we filter out those review reports prior to our case studies. In more detail, we exclude the earliest (oldest) 10% from the two datasets.Based on our observations, most of the data points falling in this range have insufficient beneficent information for the reviewer's agreement's calculation. Finally, the review reports used in this study are 55,523 review reports in Qt project and 56,038 review reports in OpenStack.

[3] `http://qt.digia.com/`
[4] `http://www.openstack.org/`

4 CASE STUDY RESULTS

4.1 *RQ1: How often does a reviewer disagree with a review conclusion?*

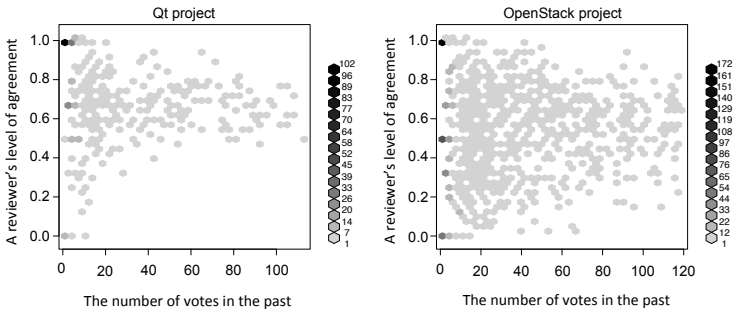

Fig. 3. The relationship between a reviewer's level of agreement and the number of votes.

Figure 3 shows the relationship between a reviewer's level of agreement and the number of votes in the past using a Hexagon binning plot. We define a reviewer having less number of votes than the median of the total votes as a *less experienced reviewer*, and one having higher number of votes than this median value as a *more experienced reviewer*, where the median values of the number of votes in Qt and OpenStack are 16 and 60 votes, respectively. From this figure, we found that *more experienced reviewers* are more likely have a higher level of agreement than *less experienced reviewers*.

Previous studies [8, 14] suggested to identify an appropriate reviewer based on expertise; however, expertise is not necessarily associated with the agreement of a review conclusion. We therefore suggest one more criteria to be considered when choosing an appropriate reviewer based on the results of this experiment. That is, if a patch owner needs the reviewers to reach a consensus for their patches as soon as possible, we suggest that patch owner to invite reviewers with higher levels of agreement.

4.2 *RQ2: What is the impact of a reviewer with a low level of agreement in a code review?*

We begin the investigation for our RQ2 with a quantitative analysis of the reviews disagreed among reviewers in the two projects. After that, we further analyze the impact of a reviewer with a low level of agreement in terms of *Reviewing Time* and *Discussing Length*.

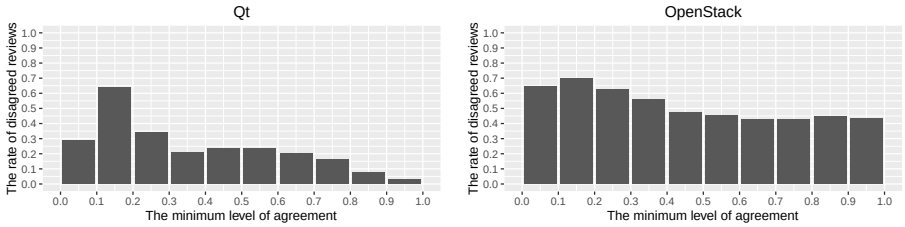

Fig. 4. The rate of reviews disagreement

Figure 4 shows the rate of the disagreed patches according to a minimum agreement of Qt and OpenStack, respectively. In both projects, when a patch owner invites a reviewer having the minimum level of agreement between 10% and 20%, we found 64% of reviews of Qt and 70% reviews of OpenStack did not reach a consensus among reviewers. Observed by correlation, we found that the rate of disagreed reviews and the minimum level of agreement exhibited strongly negative values (r=-0.79 in Qt and r=-0.90 in OpenStack). This means that a decision made by a reviewer with a lower level of agreement is less likely reach a consensus among that of the other reviewers.

Reviewing Time. Figure 5 shows that the distribution of reviewing time according to a minimum level of agreement. From this figure, in Qt project, many patches with a reviewer who has his minimum level of agreement between 70% - 90% are more likely taken much longer reviewing time than others. On the other hand, in OpenStack, the minimum level of agreement appears to be inapplicable to tell whether or not a reviewer would take a longer reviewing time. Since the features of these distributions seems to be dependent on an individual project, we therefore perform a further analysis of these two projects using a statistical method.

Figure 6 classifies the reviews into two groups, i.e., Top 50% and Remaining 50% of the population, where the cut off is determined using the median values of the minimum level of agreement of each project. In Qt, the values of the minimum, the lower quartile, the median, the upper quartile and the maximum are 0.00, 0.78, 0.83, 0.88 and 1.00, respectively, and that of the OpenStack project are 0.00, 0.40, 0.54, 0.70 and 1.00, respectively. We found that the reviews of Top 50% are likely to take shorter reviewing time than that of Remaining 50%. Confirmed by the Wilcoxon signed-rank test with p-value less than 0.01, we found that difference in the distributions between Top 50% and Remaining 50% in both Qt and OpenStack are statically significant.

Discussing Length. Figure 7 shows that the distribution of discussing length (the number of comments in a review) according to a minimum level of agreement. From this figure, in Qt, many patches with a reviewer who has his minimum level of agreement between 60% - 90% are more likely spent much longer time for their discussion than others. On the other hand, in OpenStack, many patches with a reviewer who has his minimum level of agreement less than 20% are more likely spent much longer time for their discussion than others. Similar

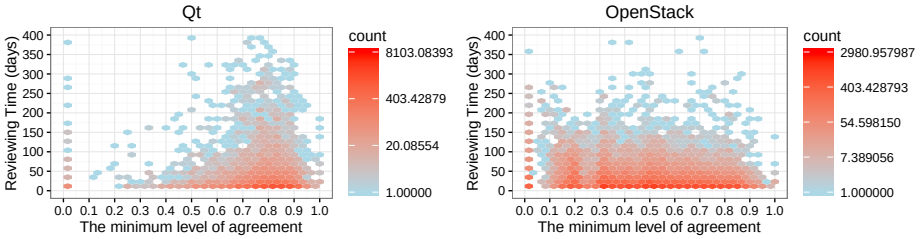

Fig. 5. The distribution of *Reviewing Time*

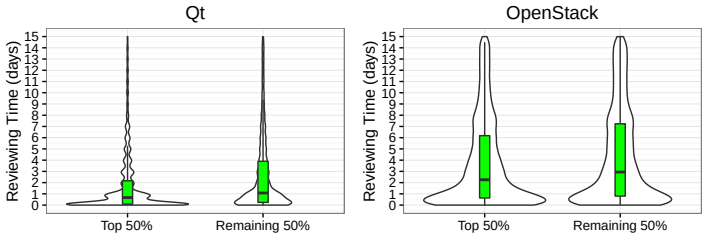

Fig. 6. The difference of the *Reviewing Time* between Top 50% and Remaining 50%

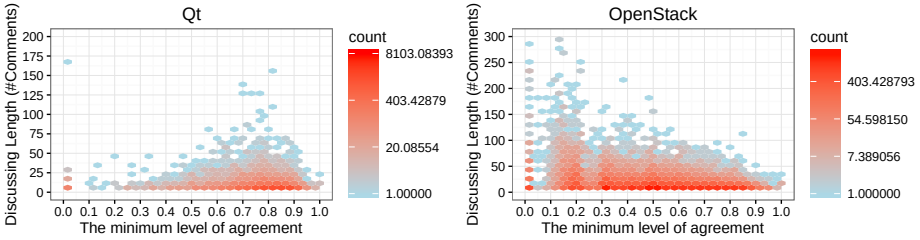

Fig. 7. The distribution of *Discussing Length*

to the *Reviewing Time*, we found that the features of these distributions also depend on an individual project. Hence, we further perform an analysis based on statistical method as done when we observed the *Reviewing Time*. Figure 8 shows that Top 50% are more likely made shorter *Discussing length*. These results were also confirmed by Wilcoxon signed-rank test as being statistically significant with p-value less than 0.01.

5 DISCUSSION

In this section, we conduct a qualitative analysis to better understand why the reviewer with a low level of agreement often disagrees with a review conclusion and makes the reviewer process become longer by reading actual review dis-

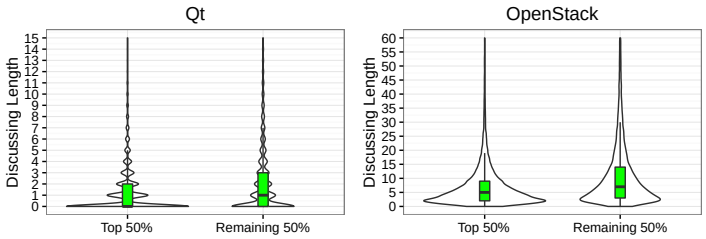

Fig. 8. The difference of the *Discussing Length*

cussions. In addition, this section also discloses the threats to validity of this study.

5.1 A Qualitative Analysis

A reviewer with a lower level of agreement is more likely overlook problems of source codes. We read an actual review report ID 8141 [5] and found that a reviewer named Claudio has overlooked the problem of the test cases, and we found that he disagreed with the final review conclusion (Abandon) of the review. To prove that our approach can explain this case, we calculate the level of agreement of this reviewer and found that the level of agreement of this reviewer was very low (classified as Remaining 50%). In addition, we also found similar cases [6] [7] as being in agreement with the analysis results of the review ID 8141.

Inviting another reviewer might make a long discussion. We read an actual review report ID 28,257 [8] and found that a record saying that reviewers assigned to this review could not complete this review and therefore asked another reviewer to verify the patches. For example, a review ID 28,257, the first reviewer having low level of agreement (classified as Remaining 50%) posted a positive vote and the second reviewer having high level of agreement (classified as Top 50%) posted a negative vote. Therefore, this discussion did not reach a consensus. In the detailed record of this review, we found that the second reviewer said that "*I recommend asking JsonDb maintainer / developers to review this change before rubber stamping it*", indicating that it seems not to be easy for the second reviewer to make a review conclusion using the first reviewer's review. In addition, we also found similar cases to this review report ID 28,257, which are [9] [10]. We therefore suggest a patch owner to invite an

[5] `https://codereview.qt-project.org/#/c/8141`
[6] `https://review.openstack.org/#/c/10363`
[7] `https://review.openstack.org/#/c/10305`
[8] `https://codereview.qt-project.org/#/c/28257`
[9] `https://codereview.qt-project.org/#/c/1048`
[10] `https://codereview.qt-project.org/#/c/7375`

appropriate reviewer (i.e., with a higher level of agreement) in order to reach a final review conclusion, with more ease.

5.2 Threats to Validity

External validity. We found a reviewer with a lower level of agreement takes longer reviewing time and makes longer discussion. We might obtain new findings if we investigate the other project with characteristics diversified from Qt and OpenStacks. Nonetheless, as our results were subject to a robust statistical significant test, we believe that the results would not be much different than what we did found in this study.
Internal validity. As we found that all the vote's scores do not always indicate the meaning of the comments, such as in some actual cases [11] [12], positive vote scores can be presented with a request to update a patch. The problem has not been investigated in this study; however, we have planned in a short future to propose an approach for its handling.

6 CONCLUSION

In this study, we investigate the impact of reviewers who have low levels of agreement using two well know OSS projects: Qt and OpenStack. In this first research question, we found that a more experienced reviewer is more likely to have a higher level of agreement than those who have less experience. In the second research question, we found that a review assigned to reviewer who has a higher level of agreement is more likely taken shorter reviewing time and spent shorter discussing length. In our further qualitative analysis, we found that a reviewer who has a lower level of agreement often overlooks problems of presented in source codes. Summarized from all the findings of this study, we suggest that a patch owner should invite an appropriate reviewer to review his patch, where the appropriateness is suggested to be determined by using the level of agreement calculated in the entire project. In the future, we will propose a more sophisticated method to invite an appropriate reviewer based on other important criteria. We believe that this future direction will be successfully contributed for a more efficient code review process.

Acknowledgment

This work has been conducted as part of our research under the Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers.

[11] `https://review.openstack.org/#/c/693`
[12] `https://codereview.qt-project.org/#/c/7242`

References

1. P. S. Kochhar, T. F. Bissyande, D. Lo, and L. Jiang, "An empirical study of adoption of software testing in open source projects," in *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, 2013, pp. 103–112.
2. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 192–201.
3. T. Hirao, A. Ihara, and K.-i. Matsumoto, "Pilot study of collective decision-making in the code review process," in *Proceedings of the Center for Advanced Studies on Collaborative Research (CASCON'15)*, 2015.
4. A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, 2013, pp. 712–721.
5. A. Bosu and J. C. Carver, "Peer code review in open source communitiesusing reviewboard," in *Proceedings of the 4th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU'12)*, 2012, pp. 17–24.
6. P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*, 2013, pp. 202–212.
7. R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, 2015, pp. 171–180.
8. P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. ichi Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, 2015, pp. 141–150.
9. K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*, ser. MSR '13, 2013, pp. 49–52.
10. M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*, 2013, pp. 45–48.
11. Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast?: Case study on the linux kernel," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*, 2013, pp. 101–110.
12. Y. Tao, D. Han, and S. Kim, "Writing acceptable patches: An empirical study of open source project patches," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME'14)*, 2014, pp. 271–280.
13. P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR'08)*, 2008, pp. 67–76.
14. X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change?: Putting text and file location analyses together for more accurate recommendations," in *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015, pp. 261–270.

15. P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, 2011, pp. 541–550.
16. A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, 2002.
17. L. Milanesio, *Learning Gerrit Code Review.* Packt Publishing, 2013.
18. P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR'15)*, 2015, pp. 168–179.