Pilot Study of Collective Decision-making in the Code Review Process

Toshiki Hirao
Graduate School of Information Science, Nara Institute of Science and Technology, Japan
hirao.toshiki.ho7@is.naist.jp

Akinori Ihara
Graduate School of Information Science, Nara Institute of Science and Technology, Japan
akinori-i@is.naist.jp

Ken-ichi Matsumoto
Graduate School of Information Science, Nara Institute of Science and Technology, Japan
matumoto@is.naist.jp

ABSTRACT

Political arguments and voting requirements often follow a simple majority method which is a decision rule that selects various alternatives that have a majority, that is, more than half the votes. On the other hand, Q&A services such as StackExchange and Yahoo! Answers use the simple majority method only as a reference point. In other words, even if an answer receives a majority vote (which represents a preferred answer for followers), the questioner might not accept the answer. In this case, the voting is used only as a reference. Open Source Software (OSS) projects might also use the voting approach only as a reference on the code review process, because a core reviewer (who makes the final decision) might not integrate a patch which was approved by reviewers into the version control system. Likewise, the votes from other reviewers might also only be used as a reference. This study identifies how many patch sets for code review follow the simple majority method in collective decision-making among reviewers. As a case study, we have analysed the criteria needed to integrate patch sets using Qt project data. From the results, we have found that only 59.5% patch sets followed the simple majority method. Patch sets with more negative votes than positive votes were likely to be rejected. Furthermore, the last vote in the first review also impacted the final decision in the first review.

Categories and Subject Descriptors

H.2.5 [**Software Engineering**]: Testing and Debugging—*Distributed debugging*; H.5.3 [**Group and Organization Interfaces**]: Collaborative computing—*supported cooperative work, Web based interaction*

General Terms

MANAGEMENT, MEASUREMENT

Keywords

Open Source Software, Code Review, Patch Submission

1. INTRODUCTION

In Open Source Software (OSS) projects, an updated source code (patch) is not always high quality and with good readability, because OSS developers (including beginners) with different technical skills contribute to creating new functions or to maintaining the products. In such a case, the patch is not always integrated into a version control system. When reviewers reject a patch, they then often request a further update or they abandon the patch entirely. Examples of the rejected patches include patches with unfixed bugs, irrelevant comments, or duplicate patches. To correctly verify the patch, OSS projects often collect votes from developers, and verify the patch thorough the collective decision-making.

A simple majority method [4] is one type of major decision rule. This method is a decision rule that selects alternatives that have a majority, that is, more than half the votes. The binary decision rule is used most often in influential decision-making bodies. Q&A services such as StackExchange and Yahoo! Answers also use the simple majority method. However, this method is used only as reference. For example, even if an answer receives a majority vote (which represents a preferred answer for followers), the questioner might choose another answer. In this case, the voting is used only as a reference. Many OSS projects also might use the same voting style only as a reference to verify a patch. If many reviewers approve a patch, the patch is likely to be integrated into a version control system. However, reviewers do not always integrate the patch. Sometimes, they reject the patch. In our experiment, we found that Qt projects rejected approximately 40% of patches, so that there was more integration than disapproved patches. It is for this reason that OSS projects might follow the minority voting depends on intensities of preferences (meaning of voting) and political equality (position of the developer in OSS project (e.g. committer)).

The goal of this study identifies how many patch sets for code review follow the simple majority method in collective decision-making among reviewers. Using Qt project data, to understand the criteria for integrating a patch in the OSS project, we analyze the final decision (to integrate or reject the patch) in the first review phase depending on the number of positive votes which approve the patch and the number of negative votes which disapprove of the patch. We target the first review phase which verifies the patch by reviewers for the first time after submitting the patch. Hence, the rejection in the first review phase includes abandonment and

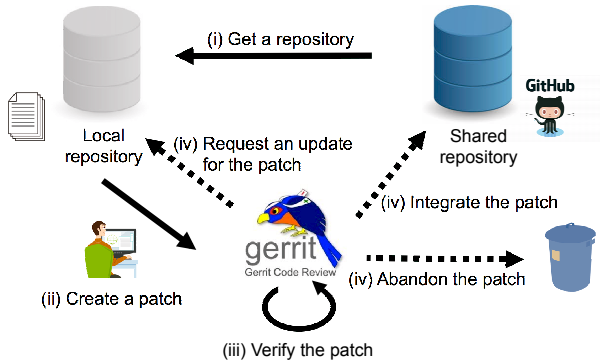

Figure 1: Overview of Modern Code Review Process

resubmission to request the patch again. Furthermore, to understand the impact of the decisive votes, we analyze the order of positive or negative votes. In the case where votes are split fifty-fifty between positive and negative votes, the reviewer would not have an easy time of making a final decision, because the reviewers would not be able to follow the simple majority method.

Contribution: The voting style based on the simple majority method is often used in many situations such as political choices. Nowadays, social network services such as Q&A services use a voting system only as a reference. In our case study, the results will aid in understanding the meaning and value of the voting style as a reference.

This paper is arranged as follows. Section 2 describes the background to this paper and related work. Section 3 provides the criteria for integrating a patch into OSS projects, Section 4 presents the impact of the voting order. Section 5 discusses our study's limitations. Finally, Section 6 concludes this paper and describes our future work.

2. CODE REVIEW PROCESS

Until recently, OSS projects have primarily used low-tech tools for patch submission and code review such as mailing lists [2, 3, 7] or issue tracking systems [9, 8] Unfortunately, developers have discussed the changed code and voted by text, because these tools are not for code review. Nowadays, there are various tools [1] that are dedicated to managing the code review process in OSS projects or projects in industry. Code review tools have helpful functions (e.g. for viewing the updated lines of code, add in-line comments in the updated lines of code, discussions and voting systems), and these functions often coordinate with version control systems such as Github. This lightweight, tool-supported style of code review found in proprietary and open source settings [6] is referred to as Modern Code Review (MCR) [1]. Figure 1 provides an overview of the Modern Code Review process. In particular, the code review process after patch submission acts as follows:

(1) One or more reviewers verify the changes proposed by the patch owner. Then, revewers will post an approving positive vote or a disapproving vote.

[1]Gerrit: `https://code.google.com/p/gerrit`, Fabricator: `http://phabricator.org/`, and ReviewBoard: `https://www.reviewboard.org/`

Table 1: Patterns of automatically-generated voting comments in the Gerrit code review system.

	score	automatic comments
positive	+2	"Looks good to me, approved"
		"Looks good to me"
	+1	"Looks good to me, but someone else must approve"
		"Works for me"
		"Sanity review passed"
		"Verified"
	0	"No score"
negative	-1	"I would prefer that you didn't submit this"
		"Sanity problems found"
	-2	"I would prefer that you didn't merge this"
		"Major sanity problems found"
		"Do not submit"

(2) Through discussion among reviewers, core reviewer decides whether or not to integrate the patch into a version control system.

The core reviewer is one of the developers who has a permit to make a final decision. He or she would make the decision based on the votes. Prior works have analyzed the acceptance for patches. Developers and reviewers should actively communicate or discuss with other team members before and after submitting their patches, which increases the chance of the patches being accepted [5, 8]. However, the decision might not always completely follow the votes. In our study, we confirm how many patches follow the simple majority method in collective decision-making among reviewers.

We conduct a case study on a large and successful OSS project. We target the Qt system, which has recorded a large amount of reviewing activity using modern code reviewing tools. *Qt* is a cross-platform application framework whose development is supported by the Digia Corporation, however, Qt welcomes contributions from the community-at-large [2]. To understand the patch decision in the modern code review process, we use the code review data of the Gerrit review databases for the Qt project shared by [5]. The databases have 70,705 review reports for Qt and contain the review status, the patch ID from the VCS repositories, votes of the reviewers, and the review comments.

We focus our analysis on those patches that are in the `merged` or `abandoned` state. To detect the votes of reviewers, we scan the comments for known patterns of automatically-generated voting comments as shown in Table 1. These votes are not only posted by an developers, but also some votes posted by automatic testing system. We target the votes posted from the automatic testing system as reviewers. Since we were not be able to detect any votes from the other fields of a review report, we filter away patches that had no reviewer comments and no reviewer votes. Finally, we targeted 66,393 review reports include the integrated 33,892 patch sets, 32,501 rejected patch sets (including 6,049 abandoned patch sets, and 26,452 re-submitted patch sets) in the first review phase.

In particular, we focused on positive votes and negative votes posted from reviewers before the final decision (Integration, Abandonment, or Resubmission) in the first review phase. In common rules of voting, the final decision would follow the majority voting. However, the number of positive votes might be the same as the number of negative votes. In this study, we analyze whether or not the final decision follows the majority voting method. In order to count the

[2]`http://qt.digia.com/`

Table 2: Distribution of the voting patterns.

	Result	$N=0$	$N=1$	$N=2$	$N=3$
$P=0$	#Integration	-	2	0	0
	#Abandonment	-	596	125	18
	#Resubmission	-	4188	322	32
$P=1$	#Integration	548	381	4	0
	#Abandonment	2,611	1,163	203	22
	#Resubmission	11,834	4,009	350	25
$P=2$	#Integration	26,239	404	1	0
	#Abandonment	583	261	31	4
	#Resubmission	3,624	778	67	5
$P=3$	#Integration	5,119	95	1	0
	#Abandonment	109	34	12	1
	#Resubmission	628	129	8	1

positive and negative votes, we define the positive votes and negative votes using a review score that was defined in the Gerrit tool. The positive votes and negative votes are shown in Table 1. The positive votes are represented by positive numbers (+1 or +2), and the negative votes by the negative numbers (-1 or -2). In this study, we do not target zero, because reviewers could not count the votes as a reference.

3. CRITERIA OF PATCH INTEGRATION

This section presents how many code reviews based on collective decision-making follow the simple majority method. These results should receive criteria to judge whether or not a patch should be integrated into a version control system. As mentioned, we analysed the final decision in the first review phase based on the number of positive and negative votes using the review repository dataset of the Qt project.

Table 2 shows that the reviewer's decision (integration, abandonment, resubmission) depends on the number of votes (positive (P) and negative (N)). #integration represents the number of merged patch sets. #abandonment represents the number of rejected patch sets. Finally, #resubmission represents the number of patch sets for which reviewers requested another update. As we mentioned in the previous section, we do not target the patch sets with without votes $\{P=0, N=0\}$, because reviewers could not count the votes as reference. Hence, we target the patch sets with less than three positive votes or less than three negative votes.

From Table 2, we found that only 59.5% of all patch sets follow the simple majority method. Those patch sets consist of integrated patch sets with more positive votes $\{\#P > \#N\}$ and the rejected patch sets with more negative votes $\{\#P < \#N\}$. These sets do not include the rejected patch sets with more positive votes $\{\#P > \#N\}$, the integrated patch sets with more positive votes $\{\#P < \#N\}$, and the patch sets with the same number of votes $\{\#P = \#N\}$. In particular, the 99.9% patch sets with more negative votes $\{\#P < \#N\}$ were rejected. On the other hand, we surprisingly found that only 61.6% patch sets with more positive votes $\{\#P > \#N\}$ were integrated.

The reviewer's decisions do not always follow the simple majority method. Some patch sets followed the simple majority method as in $\{P=2, N=0\}$, $\{P=0, N=1\}$. 86.1% ($=\frac{26{,}239}{26{,}239+583+3{,}624}$) of the cases of $\{P=2, N=0\}$ were integrated as the patch sets. Also, 99.9% ($=\frac{594+4188}{2+594+4188}$) of the cases of $\{P=0, N=1\}$ rejected as the patch sets. On the other hand, some patch sets did not follow the majority voting from reviewers, such as $\{P=2, N=1\}$. Though we assumed that $\{P=2, N=1\}$ would be integrated, 72.0% ($=\frac{261+778}{404+261+778}$) of the cases rejected the patch sets.

The patch sets with more than one negative votes were likely to be rejected. As we mentioned, there were only 61.6% patch sets with more positive votes $\{\#P > \#N\}$ were integrated. In addition, when the number of positive votes is same as the number of negative votes, $\{P = N\}$, it would be difficult for reviewers to judge whether or not the patch sets should be integrated. However, the patch sets of most cases ($\{P=1, N=1\}$, $\{P=2, N=2\}$, or $\{P=3, N=3\}$) were rejected. Indeed, 93.1% in $\{P=1, N=1\}$, 98.9% in $\{P=2, N=2\}$ and 100% in $\{P=3, N=3\}$ were rejected.

4. IMPACT OF DECISIVE VOTES

In the section 3, we found that some patch sets do not always follow the simple majority method. Also, the patch sets with more than one negative votes are likely to be rejected. This section shows which review processes lead to those final decisions (integration, abandonment resubmission) in the first review phase. To conduct on this analysis, we target patch sets with 2 or 3 reviewers' votes. Table 3 shows the final decision in patch sets with 2 reviewers' votes. Table 4 shows the final decision with patch sets with 3 reviewers votes. For example, in the Table 4, P P N means that the first reviewer posted a positive vote, the second reviewer posted a positive vote, and the third reviewer finally posted a negative vote. In this case we found 894 patch sets in this case.

In patch sets with 2 reviewers' votes, there were 4 voting patterns as shown in Table 3. Patch sets with the same kind of votes (positive or negative) by both reviewers followed the simple majority method. Indeed, 86.2% patch sets of {P P} were integrated. Also, 100% patch sets of {N N} with negative votes by both reviewers were rejected. On the other hand, another pattern of patch sets appeared with different kinds of votes by different reviewers such as {P N}, {N P}. 99.5% of {P N} were rejected and 55.5% of {N P} were integrated. Core reviewer seemed to make a decision based on the second reviewer's vote.

In patch sets with 3 reviewers' vote, there were 8 voting patterns shown in Table 4. The patch sets with 3 reviewers' votes had the majority, because more than 2 reviewers had to posts a vote which was either positive or negative. In this case, reviewers ideally followed the simple majority method. However, as mentioned before, patch sets do not always follow the method such as in $\{P=2, N=1\}$. From Table 4, we surprisingly found that less patch sets of only {P P N} followed the simple majority method. 95.4% patch sets of {P P N} were rejected. On the other hand, more patch sets of {P N P} and {N P P} followed the simple majority method. Next, we focus on the patch sets with different votes between the first reviewer and the second reviewer such as {P N P}, {N P P}, {P N N} and {N P N}. In this case, it might be difficult for the core reviewer to make a final decision on these patch sets before a third reviewer votes. However, these patch sets seemed to make a decision based on the third reviewer's vote, even if it did not follow the majority votes.

In sum, the latter votes are likely to lead the final decision to merge or reject the patch sets in the first review phase. We read some actual review reports manually and found that the first reviewer's vote was posted automatically by tests

Table 3: Final decision in patch sets with 2 reviewers' votes.

	Pattern	#Integration	#Abandonment	#Resubmission	Total
{P=2,N=0}	P P	26,293 (86.2%)	583 (1.9%)	3,624 (11.8%)	30,500
{P=1,N=1}	P N	18 (0.3%)	1,130 (23.0%)	3,751 (76.5%)	4,899
	N P	363 (55.5%)	33 (5.0%)	258 (39.4%)	654
{P=0,N=2}	N N	0 (0.0%)	125 (27.9%)	322 (72.0%)	447

Table 4: Final decision in patch sets with 3 reviewers' votes.

	Pattern	#Integration	#Abandonment	#Resubmission	Total
{P=3,N=0}	P P P	5119 (87.4%)	109 (1.8%)	628 (10.7%)	5,856
{P=2,N=1}	P P N	40 (4.4%)	219 (24.4%)	635 (71.0%)	894
	P N P	96 (43.6%)	37 (16.8%)	87 (39.5%)	220
	N P P	268 (81.4%)	5 (1.5%)	56 (17.0%)	329
{P=1,N=2}	P N N	0 (0.0%)	186 (37.8%)	305 (62.1%)	491
	N P N	1 (1.7%)	16 (28.5%)	39 (69.6%)	56
	N N P	3 (30.0%)	1 (10.0%)	6 (60.0%)	10
{P=0,N=3}	N N N	0 (0.0%)	18 (36.0%)	32 (64.0%)	50

in many review reports [3]. That is, many patch sets were decided based on the final decision made by the developer's vote.

5. THREATS TO VALIDITY

We focused our study on positive votes and negative votes posted by reviewers and by automatic testing. Those votes by reviewers have several meanings. For example, a negative vote means that the patch should not be integrated, or the patch should be fixed again. Our next major challenge is to understand the actual meaning of the votes automatically from the reviewers' discussions.

Also, we have not analyzed who posts the vote. In really, an experienced developer's vote should have a higher impact in deciding patch integration than the other reviewers. The core reviewer most likely would give a fair evaluation of the votes among reviewers. Our another challenge is to analyze who posts the vote.

Finally, in this case study, we focused only on the first review phase and one large OSS project as the pilot study. To identify the validity of our results, we should conduct this analysis on whole review phases using dataset from the other projects as well.

6. CONCLUSION

In this paper, we identified only 59.5% of code reviews that followed the simple majority method through collective decision-making. Also, we found that the reviewer's decision s do not always follow the method. Hence, we confirmed that the reviewers used the votes only as a reference. Furthermore, we found that the core reviewer might agree with the final votes in first review phase, even if they do not follow the simple majority method. In the future work, we would like to study how reviewers' votes lead to make a decision for patch sets. Furthermore, we build a prediction model to make a decision for patch sets automatically based on the voting approach.

7. ACKNOWLEDGMENTS

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.

[3]e.g. `https://codereview.qt-project.org/#/c/11751`

8. REFERENCES

[1] 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)*, pages 712–721, 2013.

[2] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in oss projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR'07)*, pages 26–29, 2007.

[3] 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)*, pages 101–110, 2013.

[4] K. O. May. A set of independent necessary and sufficient conditions for simple majority decisions. *Econometrica*, 20(4):680–684, 1952.

[5] 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)*, pages 192–201, 2014.

[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)*, pages 202–212, 2013.

[7] 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)*, pages 541–550, 2011.

[8] 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)*, pages 271–280, 2014.

[9] 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)*, pages 67–76, 2008.