

Code Reviews with Divergent Review Scores: An Empirical Study of the OpenStack and Qt Communities

Toshiki Hirao*, Shane McIntosh†, Akinori Ihara‡, Kenichi Matsumoto§

*dTosh Inc., Japan; toshiki.hirao@dtosh.com

†University of Waterloo, Canada; shane.mcintosh@uwaterloo.ca

‡Wakayama University, Japan; ihara@wakayama-u.ac.jp

§Nara Institute of Science and Technology, Japan; matumoto@is.naist.jp

I. ABSTRACT

Code review is widely considered a best practice for software quality assurance [1]. The Modern Code Review (MCR) process—a lightweight variant of the traditional code inspection process [2]—allows developers to post patches for review. Reviewers (i.e., other team members) are either: (1) appointed automatically based on their expertise [3]–[5]; (2) invited by the author [3], [6], [7]; or (3) self-selected by broadcasting a review request to a mailing list [8]–[10].

Reviewer opinions about a patch may differ. Divergent reviews can slow integration processes down [9] and can create a tense environment for contributors [11]. For instance, consider review #12807 from the QTBASE project.¹ The first reviewer approves the patch for integration (+2). Afterwards, another reviewer blocks the patch from being integrated with a strong disapproval (-2), arguing that the scope of the patch must be expanded before integration could be permitted. Those reviewers who provided divergent scores discussed whether the scope of the patch was sufficient for five days, but an agreement was never reached. One month later, the patch author abandoned the patch without participating in the discussion. Despite making several prior contributions, this is the last patch that the author submitted to the QTBASE project.

We set out to better understand patches with divergent review scores and the process by which integration decisions are made. To do so, we analyze the large and thriving OPENSTACK and QT communities. Through quantitative analysis of 49,694 reviews, we address the following research questions:

(RQ1) How often do patches receive divergent scores?

Motivation: Review discussions may diverge among reviewers. We first set out to investigate how often patches with divergent review scores occur.

Results: Divergent review scores are not rare. Indeed, 15%–37% of the studied patch revisions that receive review scores of opposing polarity.

(RQ2) How often are patches with divergent scores eventually integrated?

Motivation: Given that patches with divergent scores receive both positive and negative scores, making an integration decision is not straightforward. Indeed, integration decisions do not always follow a simple

majority rule [12]. We want to know how often these patches are eventually integrated.

Results: Patches are integrated more often than they are abandoned. For example, patches that elicit positive and negative scores of equal strength are eventually integrated on average 71% of the time. The order in which review scores appear correlates with the integration rate, which tends to increase if negative scores precede positive ones.

(RQ3) How are reviewers involved in patches with divergent scores?

Motivation: Patches may require scores from additional reviewers to arrive at a final decision, imposing an overhead on development. In reviews with divergent scores, we set out to study (a) if additional reviewers are involved; (b) when reviewers join the reviews; and (c) when divergence tends to occur.

Results: Patches that are eventually integrated involve one or two more reviewers than patches without divergent scores on average. Moreover, positive scores appear before negative scores in 70% of patches with divergent scores. Reviewers may feel pressured to critique such patches before integration (e.g., due to lazy consensus).² Finally, divergence tends to arise early, with 75% of them occurring by the third (QT) or fourth (OPENSTACK) revision.

To better understand divergent review discussions, we qualitatively analyze: (a) all 305 of the patches that elicit strongly divergent scores from members of the core development teams; (b) a random sample of 630 patches that elicit weakly divergent scores from contributors; and (c) a random sample of 305 patches without divergent scores. In doing so, we address the following research questions:

(RQ4) What drives patches with divergent scores to be abandoned?

Motivation: In RQ2, we observe that 29% of the studied patches with divergent scores are eventually abandoned. Since each patch requires effort to produce, we want to understand how the decision to abandon patches with divergent scores is reached.

¹<https://codereview.qt-project.org/#/c/12807/>

²<https://community.apache.org/committers/lazyConsensus.html>

Results: Abandoned patches with strong divergent scores more often suffer from *external* issues than patches with weakly divergent scores and without divergent scores do. These external issues most often relate to release planning and the concurrent development of solutions to the same problem.

(RQ5) What concerns are resolved in patches with divergent scores that are eventually integrated?

Motivation: In the 71% of patches with divergent scores that are eventually integrated (see RQ2), the reviewer concerns are being addressed. We set out to study which types of concerns are typically addressed.

Results: In OPENSTACK and NOVA, reviewer concerns are more often indirectly addressed (e.g., through integration timing) in patches with strong divergent scores than patches with weakly divergent and without divergent scores. On the other hand, in QTBASE, reviewer concerns are often directly addressed through patch revision, irrespective of whether divergent scores are present.

Based on our results, we suggest that: (a) software organizations should be aware of the potential for divergent review discussion, since patches with divergent scores are not rare and tend to require additional personnel to be resolved; (b) automation could relieve the burden of reviewing for external concerns; and (c) authors should note that even the most divisive patches are often integrated through constructive discussion, integration timing, and careful revision.

REFERENCES

- | Criterion | Response |
|--|---|
| The associated accepted journal paper was accepted to a journal from the list below no earlier than November 1st, 2019 and no later than November 1st, 2020: | The article was accepted for publication in the IEEE Transactions on Software Engineering on February 25th, 2020. |
| The paper is in the scope of the conference. | The paper fits under the following topics of interest that appear in the ICSE 2021 call for papers: “Mining software repositories” and “Evolution and maintenance”. |
| The paper reports completely new research results and/or presents novel contributions that significantly extend and were not previously reported in prior work. | Yes, this paper makes several novel contributions (see Section 1 of the paper for an overview). |
| The paper does not extend prior work solely with additional proofs or algorithms (or other such details presented for completeness), additional empirical results, or minor enhancements or variants of the results presented in the prior work. | This paper is not an extended version of our prior work or that of others. |
| The paper has not been presented at, and is not under consideration for, journal-first programs of other conferences. | ICSE 2021 would be the first conference at which this work would be presented. |
| The paper should not exclusively report a secondary study, e.g., systematic reviews, mapping studies, surveys. | As far as we know, this paper is not an exclusion from any of other studies. |
| [1] K. E. Wiegers, <i>Peer Reviews in Software: A Practical Guide</i> . Addison-Wesley Longman Publishing Co., Inc., 2002. | |
| [2] M. E. Fagan, “Design and code inspections to reduce errors in program development,” <i>IBM Systems Journal</i> , vol. 15, no. 3, pp. 182–211, 1976. | |
| [3] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. Matsumoto, “Who should review my code? a file location-based code-reviewer recommendation approach for modern code review,” in <i>Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering</i> , 2015, pp. 141–150. | |
| [4] M. Zanjani, H. Kagdi, and C. Bird, “Automatically recommending peer reviewers in modern code review,” <i>Transactions on Software Engineering</i> , vol. 42, no. 6, pp. 530–543, 2015. | |
| [5] M. M. Rahman, C. K. Roy, and J. A. Collins, “Correct: Code reviewer recommendation in github based on cross-project and technology experience,” in <i>Proceedings of the 38th International Conference on Software Engineering</i> , 2016, pp. 222–231. | |
| [6] 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 <i>Proceedings of the 11th Working Conference on Mining Software Repositories</i> , 2014, pp. 192–201. | |
| [7] T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K. Matsumoto, “The impact of a low level of agreement among reviewers in a code review process,” in <i>Proceedings of the 12th International Conference on Open Source Systems</i> , 2016, pp. 97–110. | |
| [8] P. C. Rigby, D. M. German, and M.-A. Storey, “Open source software peer review practices: a case study of the apache server,” in <i>Proceedings of the 30th International Conference on Software Engineering</i> , 2008, pp. 541–550. | |
| [9] P. C. Rigby and M.-A. Storey, “Understanding broadcast based peer review on open source software projects,” in <i>Proceedings of the 33rd International Conference on Software Engineering</i> , 2011, pp. 541–550. | |
| [10] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, “Communication in open source software development mailing lists,” in <i>Proceedings of the 10th Working Conference on Mining Software Repositories</i> , 2013, pp. 277–286. | |
| [11] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, “Modern code review: A case study at google,” in <i>Proceedings of the 40th International Conference on Software Engineering, Software Engineering in Practice track (ICSE SEIP)</i> , 2018. | |
| [12] T. Hirao, A. Ihara, and K. Matsumoto, “Pilot study of collective decision-making in the code review process,” in <i>Proceedings of the Center for Advanced Studies on Collaborative Research</i> , 2015, pp. 248–251. | |

Satisfying the criteria for journal first presentation at ICSE2021.