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

Toshiki Hirao, *Student Member, IEEE,* Shane McIntosh, *Member, IEEE,* Akinori Ihara, *Member, IEEE,* and Kenichi Matsumoto, *Member, IEEE*

Abstract—Code review is a broadly adopted software quality practice where developers critique each others' patches. In addition to providing constructive feedback, reviewers may provide a score to indicate whether the patch should be integrated. Since reviewer opinions may differ, patches can receive both positive and negative scores. If reviews with divergent scores are not carefully resolved, they may contribute to a tense reviewing culture and may slow down integration.

In this paper, we study patches with divergent review scores in the OpenStack and Qt communities. Quantitative analysis indicates that patches with divergent review scores: (1) account for 15%–37% of patches that receive multiple review scores; (2) are integrated more often than they are abandoned; and (3) receive negative scores after positive ones in 70% of cases. Furthermore, a qualitative analysis indicates that patches with strongly divergent scores that: (4) are abandoned more often suffer from external issues (e.g., integration planning, content duplication) than patches with weakly divergent scores and patches without divergent scores; and (5) are integrated often address reviewer concerns indirectly (i.e., without changing patches).

Our results suggest that review tooling should integrate with release schedules and detect concurrent development of similar patches to optimize review discussions with divergent scores. Moreover, patch authors should note that even the most divisive patches are often integrated through discussion, integration timing, and careful revision.

Index Terms—Modern Code Review, Divergent discussion, Empirical Study.

1 Introduction

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*, i.e., sets of changes to the software system, for review. Reviewers (i.e., other team members) are either: (1) appointed automatically based on their expertise [3], [4], [5]; (2) invited by the author [3], [6], [7]; or (3) self-selected by broadcasting a review request to a mailing list [8], [9], [10].

Reviewer opinions about a patch may differ. Therefore, patches that are critiqued by more than one reviewer may receive scores both in favour of and in opposition to integration. In this paper, we focus on these patches as their discussions have the potential to be divergent. In order to arrive at a final decision about these patches, the divergent positions of reviewers will likely need to be resolved.

Broadly speaking, divergent opinions have been studied for a long time in academic settings [11]. Recently, divergence in reviews has been studied within open source communities [12], and has been observed to correlate with negative development outcomes [12]. For example, divergence has been associated with increased rates of developer abandonment [13] and poorer team performance [14].

Specifically, in the context of code review, divergent review scores may forebode disagreement among reviewers. Divergent reviews can slow integration processes down [7], [9] and can create a tense environment for contributors [15]. 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.

In this paper, we set out to better understand patches with divergent review scores and the process by which an integration decision is made. To achieve our goal, we analyze the large and thriving OpenStack and Qt communities. Through quantitative analysis of 49,694 reviews, we address the following two 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

- *Toshiki Hirao and Kenichi Matsumoto are with the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. E-mail: hirao.toshiki.ho7, matumoto@is.naist.jp*
- *Shane McIntosh is with the Department of Electrical and Computer Engineering, McGill University, Canada. E-mail: shane.mcintosh@mcgill.ca*
- *Akinori Ihara is with the Faculty of System Engineering, Wakayama University, Japan. E-mail: ihara@sys.wakayama-u.ac.jp*

Manuscript received date; revised date.

[1] https://codereview.qt-project.org/#/c/12807/

integration decision is not straightforward. Indeed, integration decisions do not always follow a simple majority rule [16]. 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. We set out to study in reviews with divergent scores (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.
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 cases where patches with divergent scores are eventually integrated (see RQ2), the concerns of the negative score are resolved. We want to investigate which types of concerns are addressed in such cases.
Results: In OPENSTACK 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.

Our results suggest that: (a) software organizations should be aware of the potential for divergence, since patches with divergent scores are not rare and tend to involve additional personnel; (b) automation could relieve the burden of reviewing for external concerns, such as release scheduling and duplicate solutions to similar problems; and (c) authors should take heart from the fact that even the most divisive patches are often integrated through constructive discussion, integration timing, and careful revision.

Enabling replication. To enable future studies, we have made a replication package available online,[3] which includes the collected data and analysis scripts, as well as more detailed hierarchies of the reasons for abandonment and integration of patches with divergent scores.

Paper organization. The remainder of this paper is organized as follows. Section 2 describes the design of our case study, while Sections 3 and 4 present the quantitative and qualitative results, respectively. Section 5 discusses the broader implications of our results. Section 6 discloses threats to the validity of our study. Section 7 surveys related work. Finally, Section 8 draws conclusions.

2 CASE STUDY DESIGN

In this section, we describe the subject systems, their Gerrit-based review processes, and our data preparation approach.

2.1 Studied Projects

To address our research questions, similar to prior work [17], [18], we perform an empirical study on large, thriving, and rapidly evolving open source systems with globally distributed teams. Due to the manually intensive nature of our qualitative analysis, we choose to select a sample of four software projects from two communities. We select the OPENSTACK and QT communities for analysis because they have made a serious investment in code review for several years, having adopted the Gerrit tool for managing the code review process.[4] These communities are theoretically sampled [19] to represent different sub-populations. OPENSTACK is a free and open source software platform for cloud computing that is primarily implemented in Python and is developed by many well-known software companies, e.g., IBM, VMware, and NEC [3]. QT is a cross-platform application and UI framework that is primarily implemented in C++ and is developed by the Digia corporation, but welcomes contributions from the community at large [20].

The OPENSTACK and QT communities are composed of several projects. We study the two most active projects in each community (based on the number on patch submissions). Table 1, which provides an overview of the studied projects, shows that 18% (NOVA), 20% (NEUTRON), 67% (QTBASE), and 84% (QT-CREATOR) of patches involve only one reviewer. The discrepancy between the communities (OPENSTACK vs. QT) is likely reflective of differences in

[2] https://community.apache.org/committers/lazyConsensus.html

[3] https://figshare.com/s/4d3e096f4339f6d4b3da

[4] https://code.google.com/p/gerrit/

TABLE 1
An overview of the subject systems. The white and gray rows show the overview of each subject system and each community, respectively.

Product	Scope	Studied Period	#Patches	#Multi-Rev Patches	#Revs
OpenStack		07/2011 to 01/2018	444,582	297,961	9,108
Nova	Provisioning management	09/2011 to 01/2018	25,901	21,224	1,602
Neutron	Networking abstraction	07/2013 to 01/2018	12,350	9,936	1,063
Qt		05/2011 to 01/2018	168,066	36,752	1,785
QtBase	Core UI functionality	05/2011 to 01/2018	37,974	12,459	627
Qt-creator	Qt IDE	05/2011 to 01/2018	37,587	6,075	278

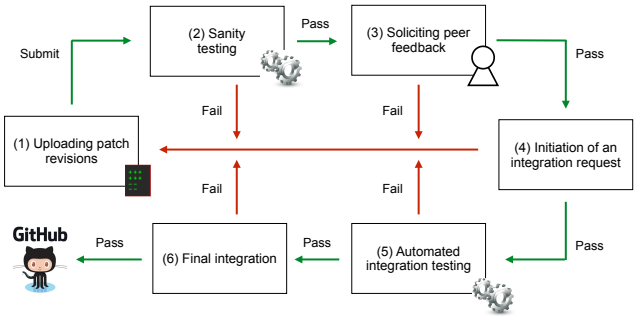

Fig. 1. The OpenStack and Qt code contribution processes.

integration policies—while OpenStack changes normally require two +2 scores for integration approval, only one +2 score is required for Qt changes.

2.2 The OpenStack and Qt Code Review Processes

The OpenStack and Qt communities manage code contributions (i.e., patches) using a rigourous modern review process that is enabled by Gerrit. Figure 1 provides an overview of that code review process, which is made up of the following steps.

(1) Uploading patch revisions. A patch author uploads a new patch or revision to Gerrit and invites a set of reviewers to critique it by leaving comments for the author to address or for review participants to discuss. Reviewers are not notified yet about the patch's details.

(2) Sanity testing. Before reviewers examine the submitted patches, sanity tests verify that the patch is compliant with the coding and documentation style guides, and does not introduce obvious regression in system behaviour (e.g., compilation error). If the sanity tests report issues, the patch is blocked from integration until a revision of the patch that addresses the issues is uploaded.

(3) Soliciting peer feedback. After a submitted patch passes sanity testing, reviewers examine the patch. The reviewer is asked to provide feedback and a review score, which may be: +2 indicating an approval for integration, +1 indicating agreement without integration approval, 0 indicating abstention, −2 blocking integration, or −1 indicating disagreement without blocking integration. Only those reviewers on the core team have access to +2 or −2 scores. Unlike the conference paper reviewing model, where positive and negative scores indicate the support for and opposition to a one-time acceptance decision, respectively, code review scores are submitted with the intention of iterating upon and improving the patch.

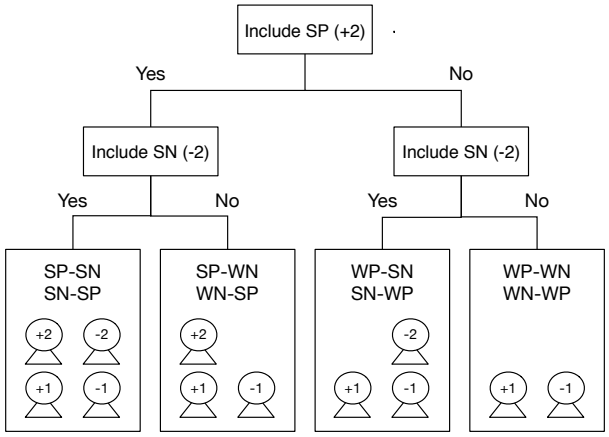

Fig. 2. An overview of our approach to classify patches with divergent scores.

(4) Initiation of an integration request. Gerrit allows teams to encode review and verification criteria that must be satisfied before patch integration. For example, the OpenStack policy specifies that a patch needs to receive at least two +2 scores.[5] After satisfying these criteria, the patch can enter the queue for integration.

(5) Integration testing. Patches that are queued for integration are checked by a set of integration tests—a more rigourous check than sanity testing. If the integration tests report issues, the patch is blocked from integration until the patch author uploads a revision of the patch that addresses the issues. These revisions should be reviewed again (if only briefly) because the fixes for integration testing issues may introduce other problems.

(6) Final integration. Once the patch passes integration testing, the patch is integrated into the upstream (official) project repositories. This step still may fail due to conflicts that may arise due to integration timing.

2.3 Data Preparation

Using the Gerrit API, we extracted review records for all of the available OpenStack and Qt patches in January 2018. Then, as described below, we prepared the data for analysis.

Divergent score identification. We begin by identifying *patches with divergent scores*, i.e., patches elicit at least one positive and one negative score. Since these patches receive opposing scores, they indicate the potential for divergence. However, patches with divergent scores may include false positive cases where there is no actual divergence between positive and negative scores. In this study, we further identify patches that have actual divergent scores due to a variety of reviewer concerns.

First, to identify whether or not a patch has divergent review scores, we select the patches that have at least one positive and one negative review score in the same revision. In cases where a reviewer updates their score, we select their

[5]http://docs.openstack.org/infra/manual/developers.html#project-gating

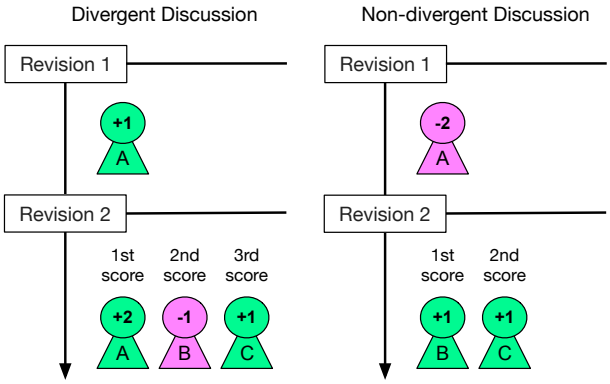

Fig. 3. Examples of review score patterns that we consider divergent and non-divergent.

TABLE 2
The number and proportion of each potentially divergent pattern in the four subject systems.

System	*SP-SN*	*SN-SP*	*SP-WN*	*WN-SP*	*WP-SN*	*SN-WP*	*WP-WN*	*WN-WP*	Total
NOVA	192 (1%)	52 (< 1%)	2051 (10%)	785 (4%)	299 (1%)	30 (< 1%)	2945 (14%)	349 (2%)	6703 (32%)
NEUTRON	112 (1%)	24 (< 1%)	918 (9%)	414 (4%)	124 (1%)	21 (< 1%)	1763 (18%)	320 (3%)	3696 (37%)
QTBASE	50 (< 1%)	11 (< 1%)	499 (4%)	302 (2%)	86 (1%)	10 (< 1%)	1163 (9%)	208 (2%)	2329 (19%)
QT-CREATOR	21 (< 1%)	4 (< 1%)	180 (3%)	81 (1%)	32 (1%)	9 (< 1%)	497 (8%)	81 (1%)	905 (15%)

TABLE 3
The integration rate of each divergent score pattern in the four subject systems.

System	*SP-SN*	*SN-SP*	*SP-WN*	*WN-SP*	*WP-SN*	*SN-WP*	*WP-WN*	*WN-WP*
NOVA	46%	94%	83%	93%	34%	50%	69%	66%
NEUTRON	58%	92%	88%	96%	19%	33%	71%	65%
QTBASE	56%	82%	81%	90%	28%	50%	73%	70%
QT-CREATOR	43%	75%	81%	96%	59%	56%	81%	90%

first score because we want to know their initial opinion before changing their mind. We then use four patterns of divergence based on the Gerrit review scores (`+2`, `+1`, `-1`, `-2`). We do not include `0` scores because they indicate no position. In addition, we consider the order in which the scores were submitted, and classify patches with divergent votes into eight patterns:

*Strong Positive, Strong Negative (*SP-SN, SN-SP*).* Patches with at least one `+2` and at least one `-2`. *SP-SN* if `+2` appears before `-2`, *SN-SP* otherwise.

*Strong Positive, Weak Negative (*SP-WN, WN-SP*).* Patches with at least one `+2` and at least one `-1`, but no `-2`. *SP-WN* if `+2` appears before `-1`, *WN-SP* otherwise.

*Weak Positive, Strong Negative (*WP-SN, SN-WP*).* Patches with at least one `-2` and at least one `+1`, but no `+2`. *WP-SN* if `+1` appears before `-2`, *SN-WP* otherwise.

*Weak Positive, Weak Negative (*WP-WN, WN-WP*).* Patches with at least one `-1` and at least one `+1`, but no `+2` or `-2`. *WP-WN* if `+1` appears before `-1`, *WN-WP* otherwise.

Figure 2 provides an overview of our classification process. For example, when classifying patch with `+2`, `-1`, and `+1` scores on the same revision, we follow the path for patches that "Include SP" on the left, and then the path for patches that do not "Include SN" on the right to arrive at the "strong positive and weak negative" category. Since the strong positive is provided earlier than weak negative, the patch is classified as *SP-WN*.

Note that the patch revision process complicates our classification approach, since review scores may be recorded with respect to different patch revisions. We focus on review scores that were applied to the same revision when identifying patches with divergent scores. For example, the scenario on the left of Figure 3 shows a patch with divergent scores, since it has at least one positive score (Rev A or Rev C) and one negative score (Rev B) on the same revision (Revision 2). Although the scenario on the right of Figure 3 also receives positive and negative scores, they do not appear in the same revision, and thus, it is not labelled as divergent.

3 QUANTITATIVE RESULTS

In this section, we present the results of our quantitative analysis with respect to RQ1–RQ3. For each question, we describe our approach for addressing it followed by the results that we observe.

(RQ1) How often do patches receive divergent scores?

Approach. To address RQ1, we examine how often the reviews of patches in NOVA, NEUTRON, QTBASE, and QT-CREATOR receive divergent scores. To do so, we compute the rate of multi-reviewer patches that receive positive and negative scores on the same revision (Table 2).

Results. Observation 1—Patches with divergent scores are not rare in OPENSTACK and QT. Table 2 shows that 15%–37% of the patches that have multiple review scores receive both positive and negative scores. While the majority of patches with divergent scores have only a weak negative score (*SP-WN*, *WN-SP*, *WP-WN*, and *WN-WP*), 7%–9% have at least one strong negative scores (*SP-SN*, *SN-SP*, *WP-SN*, and *SN-WP*) in the four subject systems (e.g., $\frac{192+52+299+30}{6{,}703} \simeq 9\%$ in NOVA).

From the alternative perspective, 63%–85% of reviews do not have divergent scores. On the surface, this may seem like reviewers often agree about patches, calling into question the necessity of having more than one reviewer evaluate patches. We caution against this interpretation because more reviewers bring additional perspectives, commenting on a variety of concerns during the review process. Moreover, reviewer scores may change across revisions—a case that we conservatively classify as non-divergent.

> 15%–37% of patches that receive multiple review scores have divergent scores.

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

Approach. To address RQ2, we examine the integration rates of patches with divergent scores. For each divergence pattern, we compute the proportion of those patches that are eventually integrated (Table 3).

Results. Observation 2—Patches that receive stronger positive scores than negative ones tend to be integrated. In patches that receive stronger positive review scores than negative ones, intuition suggests that most will be integrated. Table 3 shows that this is indeed the case, with

the *SP-WN* integration rate of 81%–88% and *WN-SP* integration rate of 90%–96% across the four subject systems. Interestingly, even patches with weakly scored criticism are not integrated on occasion, suggesting that authors are attuned to that criticism despite securing a positive score that enables integration.

Similarly, in patches that receive stronger negative scores than positive scores, intuition suggests that most will be abandoned. Again, Table 3 shows that in the NOVA, NEUTRON, and QTBASE systems, this is indeed the case in the *WP-SN* pattern, with the integration rates below 50%. The *WP-SN* integration rate is only above 50% for QT-CREATOR (59%); however, we note a greater tendency towards acceptance in all of the patterns for the QT-CREATOR system.

The results imply that a -2 score is a large obstacle to integration. Resolving criticism is a common research topic not only in code review [13], but also academic paper reviewing [11]. The prior work on code reviews suggests that when developers include constructive suggestions (e.g., alternative solutions) when critiquing patches, authors feel encouraged to continue contributing. The recommendation indicates that such a critique should be resolved through patient contributions from both reviewers and authors.

Unlike in academic peer review settings, the order in which the code review scores are recorded shares a strong relationship with the integration rate. Table 3 shows that if the negative score is submitted before the positive score (i.e., *SN-WP*), the integration rate grows by 14–22 percentage points in the NOVA, NEUTRON, and QTBASE systems. On the other hand, we observe a drop of three percentage points in the QT-CREATOR system; however, the integration rate remains above 50%.

Observation 3—Patches that receive positive and negative scores of equal strength tend to be integrated. In patches that receive positive and negative scores of equal strength, intuition suggests that half of such patches will be integrated. Indeed, Table 3 shows that the *SP-SN* pattern has integration rates of 43%–58%, which is roughly in line with our intuition. Again, the order in which scores are recorded plays a key role—when a strong positive score is provided after a strong negative one (*SN-SP*), the integration rates increase to 75%–94%, suggesting that the late arrival of a proponent can encourage contributors to revise and rescue the patch from abandonment.

Likewise, the *WP-WN* and *WN-WP* patterns have the integration rates of 69%–81% and 65%–90% respectively, which is higher than we had anticipated. The results suggest that patches with weak divergent scores are often redeemed, even if there is only weak support for them initially.

> Patches with divergent scores are often eventually integrated. The timing of the arrival of scores plays a role, with later positive scores being correlated with considerable increases in integration rates.

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

Approach. To address RQ3, we examine the review data from three perspectives. First, we report the difference in the number of reviewers in patches with divergent review scores and those without (Table 4). To analyze reviewer tendencies, we then compute the rate at which the five most active reviewers in each subject system submit scores of a different polarity than the other reviewers (Table 5). Finally, we plot the revision when divergence occurs against the last revision of each review with divergent scores (Figure 4).

TABLE 4
The number of reviewers in reviews that receive divergent scores and reviews that do not in the four subject systems.

System	*SP-SN*	*SN-SP*	*SP-WN*	*WN-SP*	*WP-SN*	*SN-WP*	*WP-WN*	*WN-WP*	Non
NOVA	4.28	4.96	3.71	4.44	3.89	3.40	3.41	3.32	3.65
	(0.63)	(1.31)	(0.06)	(0.79)	(0.24)	(-0.25)	(-0.24)	(-0.33)	–
NEUTRON	5.34	5.00	4.69	5.19	3.74	4.71	4.01	4.07	4.07
	(1.27)	(0.93)	(0.63)	(1.12)	(-0.33)	(0.65)	(-0.05)	(<0.01)	–
QTBASE	2.57	2.56	2.49	2.64	2.46	2.20	2.46	2.39	2.30
	(0.27)	(0.26)	(0.19)	(0.34)	(0.16)	(-0.10)	(0.17)	(0.10)	–
QT-CREATOR	2.44	2.00	2.24	2.42	2.21	2.00	2.26	2.22	2.17
	(0.27)	(-0.17)	(0.07)	(0.25)	(0.04)	(-0.17)	(0.09)	(0.05)	–

TABLE 5
The N→SP and P→SN rates of the five most active reviewers in the four subject systems.

Rank	NOVA		NEUTRON		QTBASE		QT-CREATOR	
	N→SP	P→SN	N→SP	P→SN	N→SP	P→SN	N→SP	P→SN
1st	17%	50%	31%	0%	50%	75%	18%	90%
2nd	58%	90%	43%	88%	36%	100%	46%	93%
3rd	49%	97%	53%	100%	35%	75%	22%	0%
4th	36%	88%	52%	90%	46%	91%	28%	0%
5rd	29%	100%	44%	84%	30%	0%	31%	83%

Results. Observation 4—Patches with divergent scores tend to require one to two more reviewers than patches without divergent scores. Table 4 shows that overall, patches that receive strong positive scores (*SP-SN*, *SN-SP*, *SP-WN*, and *WN-SP*) have a greater difference in the number of reviewers than the other patterns. Table 4 shows that specifically in *SP-SN* and *SN-SP* patterns of NOVA and NEUTRON, the differences range from 0.63 to 1.31. Unlike these patterns, the difference in patches without -2 scores (*WP-SN*, *SN-WP*, *WP-WN*, and *WN-WP*) tends to remain below one. Rigby and Bird [21] observed that two reviewers were often enough, except in cases where reviews became a group problem solving activity. Our results complement theirs, suggesting that patches with divergent scores also require additional review(er) resources to reach a final decision.

Observation 5—The most active reviewers tend to provide negative scores after positive ones. Table 5 shows that the most active reviewers provide strong negative scores after positive ones (P→SN) more often than strong positive scores after negative ones (N→SP). Indeed, 16 of the 20 analyzed reviewers stop a positive trend in review scores more often than they stop a negative trend, with personal differences of 25–72 percentage points. Active reviewers may feel pressured to critique patches with a positive trend before integration. The practice of lazy consensus, where a reviewer avoids joining a discussion when other reviewers raise concerns, may also help to explain this tendency.

In contrast, there are four reviewers who tend to stop negative trends (1st in NEUTRON, 5th in QTBASE, and 3rd and 4th in QT-CREATOR). All four of these reviewers never stop a positive trend with a strong negative score. Upon closer inspection, we find that these reviewers are senior community members who take on a managerial role. These reviewers appear to only join reviews late in order to provide a strong vote in favour for high priority content.

Observation 6—Divergent scores have a tendency to

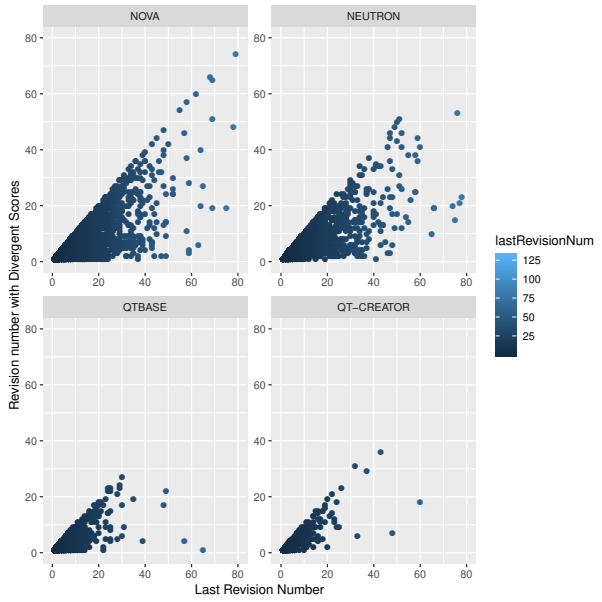

Fig. 4. The revision number that divergent scores appear on and revision number that the final decision is made in.

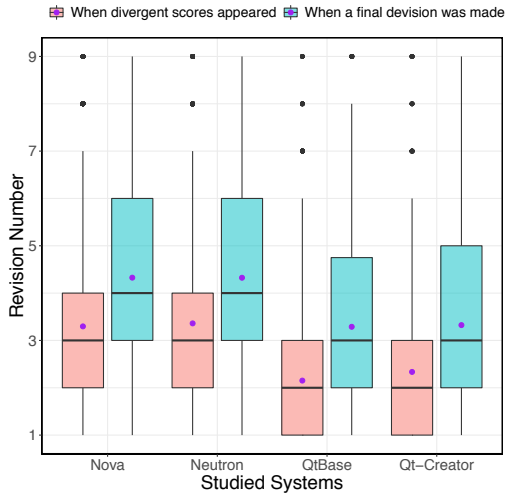

Fig. 5. The distributions of revision number that divergent scores appear on and revision number that the final decision is made in.

arise before the final revision. Figure 4 shows that the majority of divergent review scores occur well before the final revision where the integration decision is made. Moreover, Figure 5 shows that 75% of patches with divergent scores occur by the third and fourth revisions in QT and OPENSTACK, respectively. Intuitively, this may be occurring because early revisions are rougher drafts of a patch.

> Patches with divergent scores tend to require one to two more reviewers than ones without divergent scores. Moreover, the most active reviewers tend to submit negative scores after positive ones more often than vice versa. Finally, divergent scores tend to arise in the early revisions of a patch.

4 QUALITATIVE RESULTS

Our quantitative results in Section 3 (RQ1–RQ3) indicate that patches with divergent scores are not rare (Observation 1), and have a tendency to be integrated (Observations 2–4). Furthermore, on average, patches with divergent scores tend to involve one or two more reviewers than patches without divergent scores, and receive negative scores after positive ones in 70% of cases (Observations 5, 6).

In this section, we set out to better understand what concerns are raised in reviews with divergent scores through qualitative analysis. More specifically, we present the results of our qualitative analysis with respect to RQ4 and RQ5. Similar to Section 3, for each research question, we describe our approach for addressing it followed by the results.

To perform our qualitative analysis, we first analyze patches that have divergent scores in *SP-SN* and *SN-SP* patterns of NOVA and QTBASE systems. We select *SP-SN* and *SN-SP* patterns for qualitative analysis because their patches elicit strong and divergent opinions from core members of the reviewing team. Moreover, to understand the difference between patches that have divergent scores and patches that do not, we analyze a sample of patches without divergent scores that is of equal size to the number of *SP-SN* and *SN-SP* patches. The sample of those patches is selected randomly, and covers the overall studied period (i.e., September 2011—January 2018). Moreover, half of the reviewers who have participated in the *SP-SN* and *SN-SP* patches are also observed in the sample of patches without divergent scores. Finally, we analyze *WP-WN* and *WN-WP* patterns to understand disagreements that do not have integration implications (i.e., `+1` and `−1`). Since manual analysis of all *WP-WN* and *WN-WP* patterns is impractical, similar to prior work [9], we aim to achieve saturation. Like prior work [22], we continue to label randomly selected *WP-WN* and *WN-WP* patches until no new labels are discovered for 50 consecutive patches.

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

Approach. In addressing RQ4, we want to know why authors and reviewers eventually agree to abandon patches that have divergent scores. To do so, we code the review discussions of sampled patches with and without divergent scores that are eventually abandoned. Our codes focus on the key concerns that reviewers with negative scores raise. Each review may be labelled with multiple codes.

In this qualitative analysis, similar to prior studies [10], [23], [24], [25], we apply open card sorting to construct a taxonomy from our coded data. This taxonomy helps us to extrapolate general themes from our detailed codes. The card sorting process is comprised of four steps. First, we perform an initial round of coding. The first three authors were involved in the coding process such that each review was coded by at least two authors. Second, if the authors' codes disagree, the authors discussed the review in question until a consensus was reached. Third, the coded patches are merged into cohesive groups, which can be summarized with a short title. Finally, the related groups are linked together to form a taxonomy of reasons for abandonment.

TABLE 6
The reasons for abandoned patches with and without divergence.

Label	NOVA					QTBASE				
	SP-SN	*SN-SP*	*WP-WN*	*WN-WP*	Non	*SP-SN*	*SN-SP*	*WP-WN*	*WN-WP*	Non
External Concern	56 (54%)			17 (32%)	45 (41%)	14 (61%)			17 (27%)	13 (54%)
Unnecessary Fix	25	-	6	7	27	12	2	2	9	7
Integration Planning	28	2	3	3	12	1	-	2	4	1
Policy Compliance	3	-	-	-	3	-	-	1	-	3
Lack of Interest	-	-	-	-	5	-	-	-	-	3
Internal Concern	48 (46%)			36 (68%)	61 (56%)	9 (39%)			47 (73%)	8 (33%)
Design	44	1	10	19	45	6	-	18	17	6
Implementation	2	-	1	2	2	2	-	7	8	-
Testing	2	-	4	1	15	1	-	1	2	2
Works in Progress	-			-	3 (3%)	-			-	3 (13%)
Early Feedback	-	-	-	-	3	-	-	-	-	3

TABLE 7
The rate of outsider-dominated discussions.

In patches whose scores are	NOVA	NEUTRON	QTBASE	QT-CREATOR
Strongly divergent	25%	28%	$< 1\%$	8%
Weakly divergent	28%	29%	2%	2%
not divergent	11%	14%	1%	$< 1\%$

Results. Table 6 shows an overview of the categories of key concerns that were raised in the analyzed reviews of abandoned patches with and without divergent scores. A more detailed figure is available online.[3] In Table 6, the white cells show the number of occurrences of each label, while the gray cells show the number of patches with and without divergent scores that belong to the internal or external concerns categories.

Observation 7—Patches with weakly divergent scores have a higher rate of false positives than patches with strongly divergent scores. We find that patches with weakly divergent scores are less likely to actually have a divergence in NOVA and QTBASE. More specifically, we detect 142 and 73 false positive patches among the 192 and 137 potentially *WP-WN* and *WN-WP* patches in the NOVA and QTBASE systems, respectively. In most *WP-WN* false positives, reviewers with `-1` scores raise concerns that reviewers with `+1` scores have not found, which makes reviewers wait for the next revision due to lazy consensus. In contrast, we detect only four and one false positive patches among the 107 and 24 potentially *SP-SN* and *SN-SP* patches in the NOVA and QTBASE systems, respectively.

We designate another six patches as *works in progress*, i.e., patches for which the author intends to collect early feedback. These patches are not intended for integration into the codebase. For example, in review #336921,[6] the commit message states that the patch is "Internal review purpose only." The remaining patches include divergence among reviewers through divergent scores and contain *internal concerns* about the content of the patch and *external concerns* about factors that are beyond the scope of the patch content.

Observation 8—External concerns are often raised in abandoned patches that have strong divergent scores. Surprisingly, Table 6 shows that external concerns in abandoned *SP-SN* and *SN-SP* patches appear more often than internal concerns in both of NOVA and QTBASE. Moreover, when compared with abandoned patches with weakly divergent scores and patches without divergent scores, the rates of external concerns increase by 13–22 percentage points in NOVA and 7–34 percentage points in QTBASE. Similar to prior work [9], we find that reviews within our studied context are mostly topical and relevant; however, they occasionally digress into "bike shed" discussions, i.e., strongly opinionated discussions about (irrelevant) details. For example, in review #40317,[7] a reviewer explicitly mentioned that the divergent discussion is "bike-shedding." To better understand how often review discussions in our context show signs of "bike shedding," we replicate the detection approach of Rigby and Storey [9], which focuses on review discussions that are dominated by outsiders (non-core members). Table 7 shows the rate of outsider-dominated discussions in patches with and without divergent scores. The results suggest that patches with divergent scores may be more susceptible to "bike shed" discussions in NOVA, but less so in QTBASE. Indeed, the OPENSTACK community is larger (see Table 1) and involves more organizations than QT, which may explain why more reviews are dominated by outsiders. A closer look at the relationship between "bike shedding" and divergence would be an interesting topic for future work.

External concerns are further decomposed into "Unnecessary Fix," "Integration Planning," "Policy Compliance," and "Lack of Interest" categories, which we describe below.

Unnecessary Fix accounts for the majority of external concerns. This category consists of patches whose changes are not necessary or have already been addressed in another patch. For example, in review #16715,[8] a reviewer pointed out that the issue has been already fixed by review #15726.[9]

Integration Planning occurs often in the NOVA project. The category consists of patches where the integration depends on the release schedule or the integration of another patch. For example, the author of review #262938[10] was told to "resubmit this for Newton" (a future release at the time).

Policy Compliance consists of patches that violate project change management policies. For example, in review #33157,[11] reviewers argued that the patch has an incorrect change ID, which would hinder its traceability in the future.

Lack of Interest consists of patches that the author or reviewers did not pursue until integration. For example, in review #76783,[12] (a patch without divergent scores) the author did not address the reviewer feedback. The patch was automatically abandoned after one week of inactivity.

Observation 9—Internal concerns are raised at greater or equal rates in patches with strong divergent scores and those without divergent scores. Table 6 shows that in NOVA, internal concerns are raised in patches without divergent scores ten percentage points more often than in *SP-SN* and *SN-SP* patches. In QTBASE, internal concerns are raised in roughly equal proportions in *SP-SN* and *SN-SP* patches and patches without divergent scores. Internal concerns include "Design," "Implementation," and "Testing."

Design accounts for the majority of internal concerns. This concern consists of patches that suffer from patch design issues. For example, in review #87595,[13] a reviewer suggests an alternative solution for the underlying issue.

[6] https://review.openstack.org/#/c/336921/

[7] https://codereview.qt-project.org/#/c/40317/
[8] https://codereview.qt-project.org/#/c/16715/
[9] https://codereview.qt-project.org/#/c/15726/
[10] https://review.openstack.org/#/c/262938/
[11] https://review.openstack.org/#/c/33157/
[12] https://review.openstack.org/#/c/76783/
[13] https://codereview.qt-project.org/#/c/87595/

TABLE 8
The resolution of integrated patches with and without divergence.

Label	Nova					QtBase				
	SP-SN	*SN-SP*	*WP-WN*	*WN-WP*	Non	*SP-SN*	*SN-SP*	*WP-WN*	*WN-WP*	Non
Indirectly Addressed	82 (61%)		13 (26%)		5 (6%)	17 (36%)		11 (16%)		1 (4%)
Withdrawal of Negative Score	4	6	2	11	-	9	7	3	6	1
Integration Planning	47	30	-	2	5	1	-	-	2	-
Directly Addressed	52 (39%)		37 (74%)		74 (94%)	30 (64%)		56 (84%)		25 (96%)
Design	19	1	9	6	39	6	4	14	21	3
Implementation	6	4	2	7	38	11	2	8	8	15
Testing	9	4	2	5	15	2	1	-	-	5
Unnecessary Fix	6	1	3	6	11	2	2	4	9	5
Policy Compliance	3	3	-	-	-	-	2	-	1	1
No Discussion	-		-		61	-		-		26

Implementation concerns consist of patches that break backward compatibility, introduce readability issues, and/or introduce bugs. For example, in review #83667,[14] reviewers pointed out that the removal of an API was not possible, since it would break backwards compatibility.

Testing concerns consist of patches that suffer from test design and coverage issues. In situations where a change required the addition of use cases like review #21250,[15] reviewers argue that the patch should not be integrated until the requisite tests are added.

> Abandoned patches that elicit strong divergent scores tend to suffer more from external concerns than patches with weak divergent and without divergent scores.

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

Approach. In addressing RQ5, we want to know what concerns are resolved in the patches with divergent scores that are eventually integrated. Similar to RQ4, we code the review discussions of the patches with and without divergent scores that are eventually integrated in Nova and QtBase. The codes of RQ5 focus on the key concerns of the reviewer(s) who provide strongly negative scores in those patches with divergent scores. Again, each patch may be labelled with multiple codes. Similar to RQ4, we apply open card sorting to construct a taxonomy from the codes of the integrated patches with and without divergent scores.

Results. Table 8 shows the key concerns that were addressed during the reviews of integrated patches with and without divergent scores. Again, a more detailed figure is available online.[3] In Table 8, the white cells show the number of occurrences of each label, while the gray cells show the number of patches with and without divergent scores in the indirectly or directly addressed categories.

Observation 10—Integrated patches with weak divergent scores have a higher rate of false positives than patches with strong divergent scores. Similar to Observation 7, integrated patches with weakly divergent scores are less likely to have actual reviewer disagreements, while most integrated patches with strongly divergent scores do. We detect 121 and 71 false positive patches among the 168 and 133 *WP-WN* and *WN-WP* patches in the Nova and QtBase systems, respectively. Comparatively, we only detect seven false positives among the 137 and 52 *SP-SN* and *SN-SP* patches in Nova and QtBase, respectively.

We exclude 61 and 26 patches where reviewers did not provide any comments (as No Discussion) in order to clearly show the comparison between indirectly and directly addressed categories. The concerns in the remaining patches are either *directly addressed* through revision to address issues or *indirectly addressed* without revising the patch.

Observation 11—Integrated patches that have strong divergent scores indirectly address reviewer concerns more often than ones without divergent scores. Surprisingly, more than half of the integrated *SP-SN* and *SN-SP* patches do not revise the patch to address reviewer concerns. These patches fall into "Withdrawal of Negative Score" and "Integration Planning" categories.

Withdrawal of Negative Score: In these cases, the reviewer who submits a negative score is persuaded by the author or another reviewer to remove their score. For example, in review #8724,[16] the negative reviewer suspected that the patch needed to fix additional locations in the codebase; however, another reviewer persuaded them that additional fixes were not necessary. The results complement prior work [9], suggesting that if an author can adequately explain why the submitted fixes are sufficient, the likelihood of such a divergence should be mitigated.

Integration planning: In large software projects, integration efforts follow a schedule. Patch integration may be delayed if the schedule does not permit integration when the patch has been approved. For example, in review #110797,[17] while a reviewer approves the patch, a reviewer blocked its integration, since it is too late in the release schedule. After the release was cut, the reviewer withdrew the negative review score. While this category often appears in Nova, it is a rare case in QtBase, which is likely due to differences in release engineering practices.

Observation 12—In Nova, integrated patches with strong divergent scores more often indirectly address reviewer concerns, whereas the opposite is true in QtBase. In QtBase, integrated *SP-SN* and *SN-SP* patches are highly likely to directly address design and implementation concerns through patch revisions. When compared to the abandoned *SP-SN* and *SN-SP* patches in QtBase, internal *Design*, *Implementation*, and *Testing* issues account for 39% of the abandoned *SP-SN* and *SN-SP* patches, whereas they account for 55% of the integrated *SP-SN* and *SN-SP* patches. Moreover, QtBase patches with weak divergent scores directly addressed concerns more often than Nova ones do. This indicates that while integration-related concerns tend to appear and be addressed indirectly in the OpenStack community, those concerns are more often directly addressed in the Qt community.

> Integrated patches with strong divergent scores more often indirectly address reviewer concerns than patches with weak divergent and without divergent scores.

5 Practical Suggestions

In this section, we discuss broader implications of our observations for organizations, tool developers, and authors.

[14]https://review.openstack.org/#/c/83667/
[15]https://codereview.qt-project.org/#/c/21250/
[16]https://codereview.qt-project.org/#/c/8724/
[17]https://review.openstack.org/#/c/110797/

5.1 Software Organizations

Software organizations should be aware of the potential for divergent review scores. This recommendation is especially pertinent for organizations that, like OPENSTACK, set integration criteria to require more than one approval (+2 score), since this creates more opportunities for divergent scores to arise (cf. QT in Table 1). Patches with divergent scores account for 15%–37% of multi-reviewer patches in the four subject systems (Observation 1).

Patches with divergent scores tend to require additional personnel. Patches with positive and negative scores of equal magnitude have a tendency to be integrated (Observations 2 and 3). However, those patches that are eventually integrated tend to involve one to two more reviewers than patches without divergent scores (Observation 4).

5.2 Tool Developers

Automatic detection of similar changes would reduce waste in the reviewing process. "Unnecessary Fix" (e.g., Already Fixed) accounts for the majority of external concerns in patches with strong divergent scores of NOVA and QTBASE that are eventually abandoned (Observation 8). Indeed, Sedano *et al.* [26] also found that duplicated work is a frequently occurring type of software development waste. This duplicated effort is not only a waste of author time, but given that this is a common pattern in patches with divergent scores, reviewers are also wasting time to suggest improvements for patches that will not be integrated. Ideally, patch authors should check for existing solutions or ones being actively developed before developing a solution of their own. However, as Zhou et al. [27] point out, in large, decentralized projects (like OPENSTACK and QT), it is difficult for developers to keep track of concurrent development activity. Tool support for detection of similar changes in the code reviewing interface would likely save this reviewing effort. Future research may make improvements to team synchronization tools that notify team members of issues that have already been addressed by others.

Automation of release scheduling constraints will add transparency. Patches with strong divergent scores often indirectly address or are abandoned primarily due to "Integration Planning" concerns (Observations 8 and 11). In these cases, a senior member of the reviewing team applies a "procedural −2" score to prevent unplanned patches from being integrated during release stabilization. Instead, release planning could be supported by code review automation, which would relieve the senior member of the duty of applying procedural −2 scores. This automation could also increase review process transparency for patches that are uploaded at inopportune times (e.g., during release stabilization) by stating when the project will be integrating new patches.

5.3 Patch Authors

A perfect technical solution may still receive divergent scores. For a patch to be integrated, concerns that are related to the broader context of the entire project need to be satisfied. For example, the majority of abandoned patches with strong divergent scores in NOVA suffer from non-technical issues (Observation 8). The results suggest that raising the author awareness of external concerns would help to avoid such divergence. Automating the detection of similar changes and release schedule (see suggestions for tool developers above) would likely help in this regard.

Patches with strong divergent scores are often salvaged by careful revision and constructive discussion. Patches with divergent scores are often redeemed in the end. There is a skew towards integration for most of the patterns with divergent review scores (Observations 2 and 3). Furthermore, patches with strong divergent scores are often indirectly addressed through integration planning (Observation 11), while patches with weak divergent scores are mostly directly addressed by revising design and implementation concerns (Observation 12), suggesting that many of concerns of reviewers are addressable.

6 THREATS TO VALIDITY

We now discuss the threats to the validity of our analyses.

6.1 Construct Validity

Construct threats to validity are concerned with the degree to which our analyses are measuring what we aim to study. Our qualitative analyses are focused on review discussions that are recorded in the Gerrit code review system. However, there are likely cases where reviewers discuss submitted patches in-person [28], on IRC [29], or in mailing list discussions [9], [10]. Unfortunately, there are no explicit links that connect these communication threads to patches, and recovering these links is a non-trivial research problem [30], [31], [32]. On the other hand, virtually every patch in the studied OPENSTACK and QT systems could be linked to review discussions on Gerrit.

We focus our detection on patches with opposing reviewer scores that appear in the same revision. However, there may be cases where the discussion on an early revision is related with divergence that happens in a later revision. As such, our results should be interpreted as a lower bound on the rate of patches with divergent scores. Moreover, some reviewing systems do not provide a mechanism for reviewers to vote in favour or against integration of a patch. Note that this does not mean that divergence is not possible in such systems (i.e., reviewers may still disagree with each other). However, to analyze patches with divergent scores in such a setting, our detection model would need to be updated (e.g., using NLP or sentiment analysis techniques).

6.2 Internal Validity

Internal threats to validity are concerned with our ability to draw conclusions from the relationship between study variables. We analyze patches with divergent scores that solicit at least one positive and one negative review score from different reviewers. However, there might be cases where reviewers disagree with other reviewers in the discussion without submitting opposing review scores. Natural language processing techniques (e.g., sentiment analysis) could be used to approximate the opinions of the reviewers. However, such techniques are based on heuristics and may introduce noise. Nonetheless, we plan to explore the applicability of sentiment analysis in future work.

The open card sorting approach that we adopt is subject to opinions of the coders. To mitigate the risk, similar to prior work [10], [23], [24], [25], two authors discussed and agreed about all of the labels. Moreover, we make our coding results available online[3] for others to scrutinize.

6.3 External Validity

External threats are concerned with our ability to generalize our results. We focus our study on the OPENSTACK and QT communities, since those communities have made a serious investment in code review for several years (see Section 2.1). Although we conduct a multiple case study on four subject systems from those communities, we may not be able to generalize our conclusions to other systems. Replication studies are needed to reach more general conclusions.

7 RELATED WORK

Code review is a pillar of modern quality assurance approaches. Recent work has observed that the rigour of the code review process shares a measurable link with software quality. For instance, Kemerer and Paulk [33] showed that adding design and code reviews to student projects at the SEI led to software artifacts that were of higher quality. Reviewer involvement has also been linked with post-release defect proneness [6], [18], [34], as well as the incidence of software design anti-patterns [35], and security vulnerabilities [36]. Our results complement the work, suggesting that patches with divergent scores tend to involve additional reviewers in the OPENSTACK and QT communities.

While the identification and repair of defects is important, modern code review has a much broader scope [10], [37]. For example, Bacchelli and Bird [23] found that code reviews at Microsoft aim to facilitate knowledge transfer among team members. Tao *et al.* [38] found that patch design issues like suboptimal solutions and incomplete fixes are often linked with patch rejection in the ECLIPSE and MOZILLA projects. Baysal *et al.* [39], [40] found that non-technical issues are a frequent reason for patch rejection, and are likely to receive slower review responses in the WEBKIT and BLINK projects. Hellendoorn *et al.* [41] found that unconventional code is more likely to attract developer attention and face disapproval from project maintainers. Gousios *et al.* [42] found that only 13% of pull requests are rejected due to technical reasons. Observation 8 in this paper complements these results—external concerns are a frequent motivation for the abandonment of patches with divergent scores in the OPENSTACK and QT communities.

7.1 Review Process Optimization

The usefulness of code review has been a focus of recent work. Aurum *et al.* [43] have argued that reviewers should have a deep understanding of the related source code. Bosu *et al.* [44] found that module-specific expertise shares a link with the perceived usefulness of a code review. Thongtanunam *et al.* [17] showed that modules with a larger proportion of developers without code authorship or reviewing expertise are more likely to be defect-prone. Kononenko *et al.* [45] reported that Mozilla developers believe that reviewer experience is associated with review quality.

To identify reviewers with relevant expertise, reviewer recommender systems have been proposed. For example, Balachandran [46] proposed ReviewBot, which recommends reviewers based on past contributions to the lines that were modified by the patch. Thongtanunam *et al.* [3] proposed RevFinder, which recommends reviewers based on past contributions to the modules that have been modified by the patch. More recent work has improved reviewer recommendations by leveraging past review contributions [4], technological experience and experience with other related projects [5], and the content of the patch itself [47].

Even if appropriate reviewers are identified, they may not agree about whether or not a patch should be integrated. Indeed, we find that patches with divergent review scores occur often in the OPENSTACK and QT communities.

7.2 Review Discussion Process

Recent work has analyzed the discussion process in modern code review. Jiang *et al.* [48] showed that the amount of discussion is an important indicator of whether patch will be accepted for integration into the Linux kernel. Tsay *et al.* [49] showed that patches that attract many comments are less likely to be accepted. Kononenko *et al.* [50] found that review participation metrics (e.g., the number of participants in a review) are associated with the quality of the code review process. Indeed, McIntosh *et al.* [6], [18] and Thongtanunam *et al.* [34] have argued that the amount of discussion that was generated during review should be considered when making integration decisions. Our results suggest that patches with divergent scores have a tendency to be integrated in the OPENSTACK and QT communities.

Deeper analysis has found that there are surprisingly few defects caught during code reviews. Tsay and Dabbish [51] showed that reviewers are often concerned with the appropriateness of a code solution, and often provide alternative solutions during discussion. Czerwonka *et al.* [52] found that only 15% of code reviews at Microsoft discuss and address defects. Rigby and Storey [9] showed that review discussions often tackle broader technical issues, project scope, and political issues. Mäntylä and Lassenius [53] found that 75 maintainability issues are raised for every 25 functional defects raised during code review discussions of student and industrial projects. Beller *et al.* [28] found a similar 75:25 ratio in the review comments that are addressed in industrial and open source projects. Our results complement these prior studies, indicating that abandoned patches with divergent scores often suffer from external concerns and integrated patches with divergent scores often resolve reviewer concerns without revising the code change.

In code review discussions, reviewers may disagree with the other reviewers. Hirao *et al.* showed that the final decisions of patches with divergent scores do not always follow a simple majority rule [16] and that patches with divergent scores take a longer time to review [7]. Wang *et al.* [12] found that opposing views provides benefits (e.g., knowledge sharing) to the OSS projects, whereas it also has potential to lead to negative consequences. Indeed, Huang *et al.* [13] showed that disagreements in review increase a developer's likelihood of leaving the project. Filippova and Cho [14] showed that review disagreements negatively affects the

performance of the project teams. Moreover, Thongtanunam *et al.* [34] showed that review discussions with large reviewer score discrepancies share a link with files that are defective in the future. We contribute to this growing body of knowledge by quantitatively and qualitatively analyzing code reviews in two large open source communities with a focus on how often reviews with divergent scores occur and how integration decisions are reached.

8 Conclusions

Code review is a common software quality assurance practice. During the review process, reviewers critique patches to provide authors with feedback. In theory, this provides a simple feedback loop, where reviewers provide criticism and authors update patches. In practice, this loop is complex because reviewers may not agree about a patch, with some voting in favour and others voting against integration.

In this paper, we set out to better understand patches with divergent scores. To that end, we study patches with divergent scores in the OpenStack and Qt communities, making the following observations:

- Patches with divergent scores are not rare in OpenStack and Qt, with 15%–37% of patches that receive multiple review scores having divergent scores.
- Patches with divergent scores are integrated more often than they are abandoned.
- Patches with divergent scores that are eventually integrated tend to involve one or two more reviewers than patches without divergent scores. Moreover, core reviewers provide negative scores after positive ones 70% of the time on average.
- Divergent discussion tends to arise early, with 75% of divergencies occurring by the third (Qt) or fourth (OpenStack) revision.
- Patches that are eventually abandoned with strong divergent scores more often suffer from external issues than patches with weak divergent and without divergent scores. Moreover, internal concerns are raised at greater or equal rates in patches with strong divergent scores and those without divergent scores.
- Patches that are eventually integrated with strong divergent scores indirectly address reviewer concerns more often than patches with weak divergent and without divergent scores. In Nova, integrated patches with strong divergent scores more often indirectly address reviewer concerns, whereas the opposite is true in QtBase.

Based on our results, we suggest that: (a) software organizations should be aware of the potential for divergent 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 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

[1] K. E. Wiegers, *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, 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 *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 141–150.

[4] M. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review." *Transactions on Software Engineering*, 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 *Proceedings of the 38th International Conference on Software Engineering*, 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 *Proceedings of the 11th Working Conference on Mining Software Repositories*, 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 *The 12th International Conference on Open Source Systems*, 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 *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 541–550.

[9] 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*, 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 *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 277–286.

[11] F. J. Ingelfinger, "Peer review in biomedical publication," *The American Journal of Medicine*, vol. 56, no. 5, pp. 686 – 692, 1974.

[12] J. Wang, P. Shih, C, and J. Carroll, M, "Revisiting linus's law: Benefits and challenges of open source software peer review," *International Journal of Human-Computer Studies*, pp. 52–65, May 2015.

[13] W. Huang, T. Lu, H. Zhu, G. Li, and N. Gu, "Effectiveness of conflict management strategies in peer review process of online collaboration projects," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 717–728.

[14] A. Filippova and H. Cho, "The effects and antecedents of conflict in free and open source software development," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 705–716.

[15] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *International Conference on Software Engineering, Software Engineering in Practice track (ICSE SEIP)*, 2018.

[16] T. Hirao, A. Ihara, and K. Matsumoto, "Pilot study of collective decision-making in the code review process," in *Proceedings of the Center for Advanced Studies on Collaborative Research*, 2015, pp. 248–251.

[17] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1039–1050.

[18] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[19] R. K. Yin, *Case Study Research: Design and Methods*, 6th ed. SAGE Publications, 2017.

[20] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Review Participation in Modern Code Review: An Empirical Study of the Android, Qt, and OpenStack Projects," *Empirical Software Engineering*, vol. 22, no. 2, pp. 768–817, 2017.

[21] P. C. Rigby and C. Bird, "Convergent contemporary software

peer review practices," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 202–212.

[22] F. E. Zanaty, T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "An empirical study of design discussions in code review," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 11:1–11:10.

[23] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 35th International Conference on Software Engineering*, 2013, pp. 712–721.

[24] N. Kerzazi, F. Khomh, and B. Adams, "Why do automated builds break? an empirical study," in *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, 2014, pp. 41–50.

[25] M. Shridhar, B. Adams, and F. Khomh, "A qualitative analysis of software build system changes and build ownership styles," in *Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 29:1–29:10.

[26] T. Sedano, P. Ralph, and C. Péraire, "Software development waste," in *Proceedings of the 39th International Conference on Software Engineering*, 2017, pp. 130–140.

[27] S. Zhou, Ş. Stănciulescu, O. Leßenich, Y. Xiong, A. Wąsowski, and C. Kästner, "Identifying features in forks," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, p. To appear.

[28] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 202–211.

[29] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer irc meetings in open source project," in *Proceedings of the 25th International Conference on Software Maintenance*, 2009, pp. 147–156.

[30] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts." in *Proceedings of the 32nd International Conference on Software Engineering*, 2010, pp. 375–384.

[31] 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*, 2007, pp. 26–29.

[32] Y. Jiang, B. Adams, F. Khomh, and D. M. German, "Tracing back the history of commits in low-tech reviewing environments," in *Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 51:1–50:10.

[33] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on psp data," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 534–550, 2009.

[34] 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*, 2015, pp. 168–179.

[35] 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*, 2015, pp. 171–180.

[36] A. Meneely, A. C. R. Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant, and K. Davis, "An empirical investigation of socio-technical code review metrics and security vulnerabilities," in *Proceedings of the 6th International Workshop on Social Software Engineering*, 2014, pp. 37–44.

[37] T. Baum, O. Liskin, K. Nikla, and K. Schneider, "Factors influencing code review processes in industry," in *Proceedings of the 24th International Symposium on Foundations of Software Engineering*, Nov. 2016, pp. 85–96.

[38] 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*, 2014, pp. 271–280.

[39] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The influence of non-technical factors on code review," in *Proceedings of the 20th Working Conference on Reverse Engineering*, 2013, pp. 122–131.

[40] ——, "Investigating technical and non-technical factors influencing modern code review," *Empirical Software Engineering*, vol. 21, no. 3, pp. 932–959, 2016.

[41] V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli, "Will they like this? evaluating code contributions with language models," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 157–167.

[42] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 345–355.

[43] 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.

[44] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th International Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[45] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1028–1038.

[46] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 35th International Conference on Software Engineering*, 2013, pp. 931–940.

[47] 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.

[48] 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*, 2013, pp. 101–110.

[49] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 356–366.

[50] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015, pp. 111–120.

[51] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: Evaluating contributions through discussion in github." in *Proceedings of the 22nd International Symposium on Foundations of Software Engineering*, 2014, pp. 144–154.

[52] J. Czerwonka, M. Greiler, and J. Tilford, "Code reviews do not find bugs: How the current code review best practice slows us down," in *Proceedings of the 37th International Conference on Software Engineering*, 2015, pp. 27–28.

[53] M. V. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 430–448, 2009.

Toshiki Hirao is a PhD student at Nara Institute of Science and Technology, Japan. He has been a doctoral course fellowship student (DC1) in JSPS from 2017 to present. His PhD thesis aims to improve the code review efficiency. He received his Bachelor's degree in Information Science from Osaka Kyoiku University, Japan and his Master's degree in Information Science from Nara Institute of Science and Technology, Japan. More about his work is available online at http://toshiki-hirao.jpn.org/.

Shane McIntosh is the Canada Research Chair in Software Release Engineering and an assistant professor at McGill University, where he leads the Software Repository Excavation and Build Engineering Labs (Software REBELs). He received his Ph.D. from Queen's University, for which he was awarded the Governor General's Academic Gold Medal. In his research, Shane uses empirical methods to study software build systems, release engineering, and software quality: http://rebels.ece.mcgill.ca/.

Akinori Ihara is a lecturer at Wakayama University in Japan. His research interests include empirical software engineering, open source software engineering, social software engineering and mining software repositories (MSR). His work has been published at premier venues like ICSE, MSR, and ISSRE. He received the M.E. degree (2009) and Ph.D.degree (2012) from Nara Institute of Science and Technology. More about Akinori and his work is available online at http://www.wakayama-u.ac.jp/~ihara/.

Kenichi Matsumoto is a professor in the Graduate School of Science and Technology at Nara Institute of Science and Technology, Japan. He received the Ph.D. degree in Engineering from Osaka University. His research interests include software measurement and software process. He is a fellow of the IEICE and the IPSJ, a senior member of the IEEE, and a member of the JSSST. More about Kenichi and his work is available online at http://se-naist.jp/