Code review for newcomers

Is it different?

Kovalenko, Vladimir; Bacchelli, Alberto

DOI
10.1145/3195836.3195842

Publication date
2018

Document Version
Accepted author manuscript

Published in
CHASE'18

Code review for newcomers: is it different?

Vladimir Kovalenko
Delft University of Technology
Delft, The Netherlands
v.v.kovalenko@tudelft.nl

Alberto Bacchelli
University of Zurich
Zurich, Switzerland
bacchelli@ifi.uzh.ch

ABSTRACT

Onboarding is a critical stage in the tenure of software developers with a project, because meaningful contribution requires familiarity with the codebase. Some software teams employ practices, such as mentoring, to help new developers get accustomed faster. Code review, *i.e.*, the manual inspection of code changes, is an opportunity for sharing knowledge and helping with onboarding.

In this study, we investigate whether and how contributions from developers with low experience in a project do receive a different treatment during code review. We compare reviewers' experience, metrics of reviewers' attention, and change merge rate between changes from newcomers and from more experienced authors in 60 active open source projects. We find that the only phenomenon that is consistent across the vast majority of projects is a lower merge rate for newcomers' changes.

ACM Reference Format:
Vladimir Kovalenko and Alberto Bacchelli. 2018. Code review for newcomers: is it different?. In *CHASE'18: CHASE'18:IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3195836.3195842

1 INTRODUCTION

Manual inspection of source code changes—*code review*—has become a standard in modern software development [1, 2]. Developers and managers responding to online questionnaries report software quality improvement as a major benefit of code review process [1], followed by a promotion of distribution of knowledge and expertise within teams [3].

With the increasing adoption of code review in industry comes the demand for smarter tools to support the developers with this process. Notable examples of automated tool support for code review include the integration of static analysis tools' output at review time [4] and various approaches to the identification of potentially bug-introducing changes [5]. Such tools, however, focus only on the content of the changes, without considering who is the target of the automated support. In other words, the vast majority of the automated tool support for code review does not consider the social dimension of the software development process, such as team hierarchy and experience of developers.

A notable category of developers who would benefit from tailored support is the category of new contributors, aka *newcomers*. In fact, successful contribution to software projects requires a certain degree of familiarity with the codebase [6]. Empirical research demonstrates that lack of such familiarity is commonly acknowledged as a problem by new contributors [7]. Problems with onboarding result in lower quality of newcomers' output: for example, bug density in their code is higher [8]. As a reaction, some software teams take measures to help developers integrate and climb up the learning curve faster [9], for example by providing mentorship to newcomers [10].

Mitigating newcomers' onboarding difficulties is specifically important for open source software (OSS) projects, which often encourage external contributions [11], but these projects are often limited in the resources they are able to invest in one-to-one mentoring. As an alternative, to tackle this limitation, more senior team members may review newcomers' contributions more thoroughly and with a higher priority. However, little empirical knowledge is available on whether and how this happens in practice. Knowledge is limited despite the fact that identifying newcomer-specific effects in code review process is important both (1) to obtain a deeper understanding of the onboarding process and its variation among different ecosystems and technology stacks and (2) for development of practices and tools that help teams to deal with onboarding issues more efficiently.

In this study we set out to explore the differences in code review process for changes that are authored by developers with very little track record of prior contribution.

We focus on three popular OSS ecosystems that use Gerrit for code review and perform a fine-grained analysis by considering each subproject independently. We compare reviews for newcomers' changes vs. reviews for changes by other developers along several dimensions: (1) reviewers' experience (based on developers' track record as change authors and reviewers), (2) review attention (based on metrics such as the number of unique reviewers and comments, duration of review, and time to first comment), and (3) merge rate (likelihood to merge).

The results of comparison greatly vary across three ecosystems, and even among different projects in each ecosystem. The only exception is merge rate, which is 12-19% lower for newcomers' changes in all three ecosystems. Lack of a universal effect suggests that understanding of newcomer-specific aspects of code review process requires more thorough analysis beyond straightforward quantitative methods, despite the availability of large amount of historical data.

CHASE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5725-8/18/05…$15.00
https://doi.org/10.1145/3195836.3195842

2 BACKGROUND

Existing evidence of newcomer-specific effects in code review is limited and contradictory. Bosu *et al.* [12] used social network analysis to identify core and peripheral developers in several OSS projects using Gerrit [13], and compared some characteristics of code reviews between these categories of developers. They found that peripheral developers' changes are less likely to be merged and take longer to review. Conversely, Gousios *et al.* [2] conducted a large-scale analysis of pull request lifecycles in GitHub projects and found that pull requests receive no special treatment and have almost equal chances to get merged, irrespective of whether they come from external contributors or the core team.

Apart from the contradictory results, one limitation of these studies is that they categorize developers based on their position in project's social network, rather than on their track record, thus not allowing to account for variety in project scale: peripheral developers in large projects can author hundreds of commits. In addition, Bosu's study [12] treats Gerrit instances of OSS ecosystems as whole monolithic projects, which does not reflect project organization in Gerrit workflow, where codebase is organized into multiple smaller projects. While such monilithic approach allows to account for cross-project experience of developers, it omits interesting events of sporadic contributions of tenured developers in sub-projects outside of their usual work area. Authors of such contributions are effectively newcomers, despite their possibly central position in the larger ecosystem network.

3 RESEARCH METHOD

We define three research questions to investigate relevant dimensions in which the review process for newcomers may differ:

RQ 1: Are changes from new developers reviewed by more experienced developers? Lower quality of newcomers' changes and higher need for learning can be addressed by assigning more experienced developers to review their changes. Thus, we investigate whether OSS projects adopt this practice.

RQ 2: Do changes from new developers receive more attention during review? Low experience of author of changes can lead to increased attention from reviewers, who could give suggestions and support to help the onboarding of the new contributors. This would result in longer reviews with more comments from more people. Time before first comment might be larger, indicating a possibly lower priority of newcomers' changes.

RQ 3: Are changes from new developers less likely to be merged? We can expect newcomers' changes to be merged less often, as another consequence of the time it takes new developers to become accustomed with a system and its inherent style [14].

For each RQ, we calculate a specific metric from historical data, split all observations into *newcomer-authored* and *other* groups, and compare distributions of the metric between two groups.

3.1 Setting and method

Subject systems and reviews. We select three large open source ecosystems – QT, Eclipse and OpenStack, as the subjects of our study. In each of these ecosystems, a dedicated instance of Gerrit [13] is used for code review, and the codebase is stored in multiple Git repositories, each of which corresponds to a separate *Gerrit project*. Some of the existing studies [15, 16], when focusing on open source Gerrit instances, treat the whole instance as one large project. Instead, to achieve a finer granularity level, we analyze each Gerrit project independently. Table 1 presents the total activity figures of the subject projects. For each Gerrit instance, we select 20 projects that have had the most reviews created during the last year. For each of these 20 projects, we use Gerrit API to download the review data (*i.e.*, review participants, review actions and comments) along with the VCS activity history (*i.e.*, change author and timestamp).

Table 1: Total activity in subject projects. An "event" corresponds to participation of one developer in one review as author. Event counts reported are after filtering.

	Total changes	Total reviews	Events (newcomers)	Events (total)
Eclipse	389,748	49,524	416	19,295
OpenStack	854,786	154,527	9,300	142,811
QT	635,043	126,219	3,325	108,512

We discard review events and comments performed by automatic tools, such as sanity checker bots. To count the contributions from different accounts of the same person together, we apply a simple name- and email- based author disambiguation algorithm.

For each code review in each Gerrit project, we extract every event of type *REVIEW*, which represents an approval or rejection of current changeset by a reviewer. Using the VCS activity data, for each review event we identify the authors of the changeset under review. This way, each event yields one or more *author + reviewer* pairs. Each of these pairs represents an event of an *author*'s code change being reviewed by a *reviewer*.

Using both code review and VCS activity data, we calculate the amount of prior project participation of author and reviewer for each review event. To adjust for difference in size and activity rate between Gerrit projects, we also calculate project-wide aggregated activity metrics, such as the average number of changes made or reviews performed per developer, at the point of each review.

When comparing developers' experience, we use number of their contributions relative to the average number of contributions per developer in the project, rather than the absolute count of commits or reviews. This metric changes less with the growth of a project, thus is more suitable for comparing events in different stages of a project lifespan.

Identifying newcomers. Depending on the number of previous contributions, we label some events as authored by a newcomer.

A typical distribution of commit counts per author for an open source project is right-skewed (*i.e.*, many authors do not contribute to the project continuously). Such asymmetry makes it difficult to define a newcomer by simply setting a fixed threshold on prior contributions count: For any reasonably high numeric threshold (tens of commits) most contributors would be considered newcomers. Even though it may represent reality accurately for projects that are maintained by a large community, such a definition would make little sense for this study because the edge-case effects that we want to focus on are likely to be blurred. For this reason, we use a strict definition: We consider newcomers the developers who are making their first, second, or third ever contribution to a project.

Calculating the metrics. For RQ1, we evaluate reviewers' experience in *number of past reviewed changes by a reviewer divided by average number of changes reviewed by any reviewer in project* for each reviewed changeset. To take effect of project growth into account, we normalize the number of changes by average number of changes per developer in project. It allows us to compare a reviewer to others in different stages of their affiliation with the project, adjusting to growth of absolute number of reviews. We also compare a similar metric for number of changes *authored* by a reviewer. For RQ2, we use numbers of reviewers and comments, time between review creation and first comment, and total review lifespan (time between creation and last action), as metrics of reviewers' attention. For RQ3 we calculate merge rate by dividing number of events from merged reviews by total number of events.

Comparing the metrics. Having split the observations into two sets (*i.e.*, reviews to changes authored by newcomers vs. changes authored by other developers), we evaluate the difference in the distributions of our metrics in these sets. A difference in distributions of two subsets of a metric indicates a relationship between the 'newcomer' factor and the metric. We test the hypothesis of the two distributions being shift relative to each other with Mann-Whitney U test, using a p-value of 0.01 to define significance. For comparison of review attention metrics and review merge rate, we filter the data to only leave one event per author per review, not to let multiple occurrences of events for the same author distort the distribution of review-wide metrics. These filtered numbers are reported as counts of events in Tables 1 and 3.

4 RESULTS

We report results aggregated per ecosystem. Results per project, p-values, and a reproduction package are available online.[1]

RQ1: Reviewers' experience. Table 2 presents the results of the comparison of reviewers' experience and review attention metrics between changes authored by newcomers and changes authored by other developers. The results vary between the systems. In most Eclipse projects, the comparison reveals no significant difference in metrics of reviewers' experience; the possible reason is smaller project sizes and consequent smaller sizes of distributions in comparison. For 13 of the 20 OpenStack projects, regardless of chosen experience metric, the changes from newcomers are reviewed by *less* experienced developers than changes from developers who are not newcomers. QT projects display the most diverse results: About half of the projects display no difference in experience of reviewers, but for most of the other projects newcomers' changes are reviewed by *less* experienced developers than changes from developers who are not newcomers. A few other QT projects, however, demonstrate an opposite effect.

RQ2: Reviewers' attention. In most Eclipse projects, similarly to reviewers' experience, the changes from newcomers do not differ from other changes significantly in terms of attention metrics. In most of the OpenStack projects, the number of comments and time to first comment are less for newcomers' changes; number of reviewers and review lifespan are different for newcomers' changes, but the direction of this difference is not consistent. For most of QT projects, most metrics reveal no difference between newcomers and others in most of the projects. Reviewers count for newcomers' reviews is different in 11 of 20 projects. The direction of the difference, however, is not consistent: The metric is greater for newcomers in 6 projects and less in 5 others.

RQ3: Review outcomes. Results of review merge rate comparison are presented in Table 3. Consistently across all three systems, we found newcomers' changes less likely to be eventually merged. Depending on the project, changes authored by newcomers are 12% to 19% less likely ('ratio' column in Table 3) to be merged.

Table 2: Comparison of reviewer experience and review attention metrics. Each cell counts the projects in the ecosystem with the corresponding direction of metric difference; numbers in bold when it is the majority of the projects.

		For newcomers' changes, metric is		
	Metric	higher	same	lower
Eclipse	Reviewers' exp. (past changes)	2	**16**	2
	Reviewers' exp. (past reviews)	1	**16**	3
	Comments count	3	**17**	0
	Reviewers count	0	**12**	8
	Time to first comment	1	**18**	1
	Review lifespan	0	**18**	2
OpenStack	Reviewers' exp. (past changes)	0	7	**13**
	Reviewers' exp. (past reviews)	0	7	**13**
	Comments count	1	5	**14**
	Reviewers count	6	7	7
	Time to first comment	0	6	**14**
	Review lifespan	6	9	5
QT	Reviewers' exp. (past changes)	3	**10**	7
	Reviewers' exp. (past reviews)	1	9	**10**
	Comments count	3	**12**	5
	Reviewers count	6	9	5
	Time to first comment	1	**14**	5
	Review lifespan	3	**15**	2

Table 3: Merge rates in subject ecosystems. An "event" corresponds to the participation of one developer in one review as author. Event counts are after filtering.

	Merged events		Merge rate		
	newcomers	others	newcomers	others	ratio
Eclipse	291	16,334	0.70	0.87	0.81
OpenStack	6,478	113,669	0.70	0.85	0.82
QT	2,633	94,626	0.79	0.90	0.88

5 DISCUSSION

In the majority of the most active projects in OpenStack and in about a half of the most active projects in QT, there exists an association between the author of changes being a newcomer in a project and code review process metrics.

Less expertise and attention. Newcomers' changes, when compared to others' changes, are reviewed differently in terms of *who* reviews their changes (reviewers' experience), *how* their changes

[1] http://www.ifi.uzh.ch/en/zest/research/newcomers-chase18.html

are reviewed (attention metrics), and the likelihood of their changes to be merged. Contrary to our intuition, based on the idea that differences in metrics for newcomers' changes might be associated with mitigating the effects of their low experience, we found reviewers of newcomers' changes to be more often *less* experienced in a project, than the other way around. However, strength and direction of differences in reviews for newcomers vs. others vary across the systems. This variance suggests that the difference in actions towards newcomers' changes might indicate some kind of a special attitude to newcomers that is specific for certain ecosystems or projects; alternatively, variance in effects can be attributed to variance in types of changes that are commonly submitted by newcomers. Future research should focus on exploring its nature further. The possibility of associating of peer attitude with quantitative characteristics of review activity, along with lack of solid research background in this area, suggests a demand for deeper investigation of the impact of status and other social factors on team's interaction. Such research is important for setting up a methodology for identifying social effects in software teams' work artifacts, which can find applications in next generation of team collaboration tools.

Changes' content. In our study we only focused on the quantitative metrics of code review for newcomers, without looking into contents of reviews. However, as previously mentioned, the difference in metrics could be attributed to difference in contents of the changes (*e.g.*, one may hypothesize that OpenStack's newcomers usually receive less comments because they usually make trivial changes, compare to newcomers of other ecosystems). However, this limitation is common for all quantitative studies. Quantitative methods that we used in this study are only capable of unraveling the most obvious effects. Lack of evidence of strong effects in our study suggests the need for a qualitative study to explore the newcomer-specific aspects of code review more deeply.

Estimating reviewers' experience. The metric that we use to estimate reviewers' experience could be improved. The average number of changes/reviews per developer (which we use to normalize the absolute count of contributions) does not always remain the same through the project lifespan. For example, when a project quickly gains popularity and attracts a lot of contributions from new people, the average number of contributions per developer decreases. This leads to an increase of estimated experience for existing developers, which does not represent actual gain of their experience. However, we believe that this point does not affect validity of our study, as we do not use the absolute values of this metric, but only compare developers against each other; moreover, several polar instances of this effect can even out when we compare metrics for the whole history of the project, which contains thousands of reviews. A finer method to estimate a developer's project-wide experience is a worthwile point for future work.

While the choice to treat every project independently provides finer granularity and reveals differences between the projects in an ecosystem, it does not let us consider cross-project experience of developers. People who typically contribute to related subsystems (which we considered as different projects in our study) should gain experience with a new subsystem faster thanks to their understanding of the whole system. Because cross-project connections are out of the picture, our method considers someone who contributed to other subsystems and had already gained solid experience and reputation, a newcomer, just as someone who makes the first contribution to the whole subsystem. This issue should be addressed in future work by treating the contribution history more precisely.

6 CONCLUSION

Understanding the impact of the code review process on the distribution of technical knowledge in a team, and vice versa, is vital for arming the code review tools with features to optimize the social outcomes of the process. We conducted a large-scale empirical quantitative investigation focused on the existing differences in the review process for newcomers across 60 OSS projects belonging to three ecosystems. We found that in some cases newcomers' changes are reviewed differently than other changes. Namely, the reviews differ in terms of reviewers' experience, number of comments, time before first comment, and other metrics. However, strength and direction of the differences vary across systems and projects.

This study suggests several directions for future work. Variation of effects between some projects and subsystems and their similarity among others suggests a promising course for a deeper exploration of newcomer onboarding, and its aspects that are specific to certain ecosystems.

ACKNOWLEDGMENTS

Bacchelli gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Project No. PP00P2_170529.

REFERENCES

[1] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *ICSE 2013*. IEEE Press, pp. 712–721.
[2] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *ICSE 2014*. ACM, pp. 345–355.
[3] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *ESEC/FSE 2013*. ACM, pp. 202–212.
[4] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *IEEE software*, vol. 25, no. 5, 2008.
[5] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," in *PROMISE 2009*. ACM, p. 7.
[6] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," in *CASCON First Decade High Impact Papers*. IBM Corp., 2010, pp. 174–188.
[7] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, and D. Redmiles, "The hard life of open source software project newcomers," in *Proceedings of the 7th international workshop on cooperative and human aspects of software engineering*. ACM, 2014, pp. 72–78.
[8] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess?" in *MSR 2011*. ACM, pp. 153–162.
[9] F. Fagerholm, A. S. Guinea, J. Borenstein, and J. Münch, "Onboarding in open source projects," *IEEE Software*, vol. 31, no. 6, pp. 54–61, 2014.
[10] M. Johnson and M. Senges, "Learning to be a programmer in a complex organization: A case study on practice-based learning during the onboarding process at google," *Journal of Workplace Learning*, vol. 22, no. 3, pp. 180–194, 2010.
[11] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *ICSE 2003*. IEEE, pp. 419–429.
[12] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in *ESEM 2014*. ACM, p. 33.
[13] "Gerrit code review," https://www.gerritcodereview.com/, accessed: 2018-01-31.
[14] V. J. Hellendoorn, P. T. Devanbu, and A. Bacchelli, "Will they like this?: Evaluating code contributions with language models," in *MSR 2015*. IEEE Press, pp. 157–167.
[15] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *SANER 2015*. IEEE, pp. 141–150.
[16] 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 *ICSE 2016*. ACM, pp. 1039–1050.