Code review for changes by newcomers: Is it any different?

***Abstract*—Manual inspection of new source code changes, or code review, has become a standard procedure in modern software development. Among its various benefits is the exchange of knowledge within a team. Some of code review effects, such as quality improvement, can be observed and measured. Expertise exchange within a team is harder to formalize and observe.**
Meaningful contribution to a software project requires familiarity with the codebase, so effects of experience are likely to be the most evident during the onboarding of a new developer.
In this study we investigate the effect of developers' low experience with the project on the code review process metrics. We explore the connection between experience of changes' author and review process and its outcomes, by examining if they are different for changes authored by newcomers. We compare reviewers' experience, effort metrics and change merge rate for newcomers' changes and those by more experienced people in 60 active open source projects. We find that for most of studied projects a significant difference exists in terms of reviewer experience and effort metrics. We also find that the only phenomenon that is consistent across the vast majority of projects is a lower merge rate for newcomers' changes.

***Keywords*-code review; newcomers; developer onboarding**

I. Introduction and background

The manual inspection of source code changes has become a standart in modern software development process [1], [2]. The lightweight approach used in industry is a tool-based manual review, also known as Modern Code Review (MCR). Developers and managers responding to online questionnaries report software quality improvement as a major benefit of review process [1].

Apart from straightforward ways of quality improvement such as early bug detection, the long-term benefits of MCR include improved code maintainability [3], compliance of code with conventions and guidelines [4], and exchange of technical knowledge within the team [5]. Some of existing research work is motivated by the need to make the code review process more efficient. A vital step in improvement of every process, especially as complex as human collaboration, is establishing a reliable model of the underlying interaction patterns [6]. Existing work aims at optimizing the whole code review process in terms of effort required from developers. Reviewer suggestion models [7], [8] help developers assign the right reviewers to changes, which results in faster and deeper reviews; bug prediction approaches [9], [10] suggest to allocate more code review effort to the parts of codebase that demand it more. When applied, these approaches are ultimately meant to improve the quality of code after review, given the same amount of human effort. In this work we focus on another major effect of code review that, however, does not affect the code itself – distribution of knowledge and expertise within a team.

The concept of expertise is subjective and hard to formalize, so reliable observation of the effects of code review on expertise transfer and its overall benefits of sharing knowledge for the team is a more complicated task than evaluating code-related improvements such as decrease of defects density.

In the preceding decades social science and management theory have revealed patterns, observed consistently across various environments and industries, in the way teams collaborate and individuals behave within a team. Several definitions and models for expertise of knowledge workers were suggested as well [11], [12]. Not much existing work is dedicated to applying these broader concepts to software engineering process and code review in particular.

The effects of knowledge transfer are most tangible when a new developer joins a team [13], [14]. Some software teams take measures to help developers integrate by providing mentorship to new developers [15]; the onboarding process of new team members is also described at a higher level in organizational management theory [16], [17].

In this work we identify the effects of onboarding process on code review in terms of review process metrics, expertise of participants, and outcome of the reviews. We explore if code review process and its outcomes are significantly different for changes authored by *newcomers* – developers with little or no prior participation history within the project. One goal of this study is to observe some known pattern in human interaction, that is described at a higher level, in the historical data from code review.
Another goal is to identify effects of review participants' experience on the code review process flow. This knowldege would contribute to future research regarding the role of code review process on the exchange of expertise in teams and possible ways of supporting the expertise flow with tools.

Contributions from new people are often encouraged in open source projects [18] and may make up to a significant share of all changes in project history. Even if a project is only mainained by long-term collaborators with no third-party contributors, every developer remains a newcomer for some time after they start their work on a project.

Successful participation in a software project as a developer requires a certain degree of familiarity with the codebase. Consequently, the expertise transfer and effects of participants' experience on the process are likely to be the most evidently

observable during the early stage of a developer's work on a project. Existing work in software engineering research suggests more reasons to expect the code review process to be different for newcomers. Mentoring of new developers is known to help them climb up the learning curve faster [19]. However, with often limited human resources, assigning a mentor to a new contributor is not a common practice in community-driven software projects. The difficulties a newcomer faces during onboarding may lead to increased attention from more senior developers to newcomers. Eyolfson *et al.* [20] explicitly suggest this practice: *"we may want to promote the practice of daily committing developers code-reviewing other developers commits"*. Due to complexity of software projects, lack of developers' experience may naturally lead to higher bug density in their changes [20]. One of goals of code review process is prevention of post-release defects, so this inequality is another reason to expect changes from newcomers to be reviewed more thoroughly, or by more experienced developers, than other changes.

II. Research method

A. Research questions

We defined three research questions to investigate the potential dimensions of difference in review process for newcomers. For each research question we calculated a corresponding metric from historical data, split all observations into *"newcomer-authored"* and *"other"* groups, and compared the distribution of the metric in two groups. Splitting and comparison methods are described later in this section.

RQ 1: Are changes from new developers reviewed by more experienced developers? We evaluated the 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. The average is used to take effect of project growth into account, thus allowing to compare a reviewer to others in different stages of their affiliation with the project, during which an absolute number of reviewed changes grows.

We also compared a similar metric for number of changes *authored* by a reviewer.

RQ 2: Do changes from new developers receive more attention during review? We used numbers of reviewers and comments, time between review creation and first comment, and total review lifespan (time between creation and last action), as estimators of reviewers' attention.

RQ 3: Are changes from new developers less likely to be merged? We calculated the merge rate for reviews and compared it for the two groups.

B. Setting and method

1) Subject systems and reviews: We selected three large open source ecosystems – QT, Eclipse and OpenStack, as the subject for the study. In each of them a dedicated Gerrit instance is used for code reviews.
In all the systems, the codebase is stored in multiple Git repositories, each of which is represented by a separate *Gerrit project*. Some of the existing studies focusing on open source Gerrit instances treat the whole instance as one large project [7], [21]. To achieve a finer granularity level, we analyzed each Gerrit project independently. For each Gerrit instance, we selected 20 projects with the most review activity during the last year. For each of these projects, we downloaded and processed the review data, consisting of review participants, review actions and comments, along with the VCS activity history, that included author and timestamp for every change.

We discarded automatic review events and comments such as actions by sanity checker bots. To count the contributions from different accounts of the same person together, we applied a simple name- and email- based author disambiguation algorithm. In further analysis, we represented authors and reviewers using aliases assigned during the disambiguation.

We processed both review and changes data. Iterating over all events sorted by time, we recorded the total numbers of previous contributions (authored and reviewed changes, separately) for each actor (author or reviewer of a change) by the time the change was authored. As the result, for each review we knew the amount of prior participation of author and reviewer(s) up to the point the review took place. Aggregated information on total project activity was also recorded for all reviews.

2) Identifying newcomers: Depending on numbers of previous contributions, some events were labeled as authored by a newcomer. The labeling method is explained below.

A typical distribution of commit counts per author for an open source project is right-skewed: many authors do not contribute to the project continuously.

This asymmetry implies difficulty in defining newcomers by setting a fixed threshold on prior contributions count: for any reasonably high numeric threshold (tens of commits) most project developers would be considered newcomers. Though it may represent reality quite accurately for projects collaboratively 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 considered newcomers the developers who are making their first, second, or third ever contribution to a project. As a project grows and becomes more complex, it also becomes harder for new developers to understand. Thus, a newcomer threshold, expressed in a minumum number of contributions that a developer should make before they gain certain extent of understanding of code, should naturally increase with time. To take this point into account, we devised an alternative definition of newcomer to comply with the growth requirement. We considered a newcomer every one who authored less changes than a median number of changes authored per person to date. It is important to note that, despite a median dividing all developers into two equal parts, newcomers defined by median were a minority in all analyzed projects. This effect is caused by significant share of past contributors, with changes count below median, who never contribute to a project again, thus not creating new data points.

TABLE I: Comparison of reviewer expertise metrics

System	Experience metric	Newcomer definition	Metric is greater for newcomers ***	**	*	No sig. diff.	Metric is less for newcomers *	**	***
Eclipse	Changes	Median	0	3	1	**12**	0	1	3
		<=2	1	1	0	**16**	2	0	0
	Reviews	Median	1	0	0	**11**	2	2	4
		<=2	1	0	0	**16**	1	0	2
OpenStack	Changes	Median	0	1	0	3	3	0	**13**
		<=2	0	0	0	6	1	0	**13**
	Reviews	Median	0	0	0	4	0	3	**13**
		<=2	0	0	0	6	1	1	**12**
QT	Changes	Median	4	0	1	5	2	2	6
		<=2	1	2	0	**10**	0	3	4
	Reviews	Median	2	0	0	2	1	3	12
		<=2	1	0	0	6	3	0	**10**

In further analysis, we used both dataset splitting strategies. In the tables the strategy where newcomers are those who authored *0, 1 or 2 changes* in the past are referred to as "$\leq$ 2 changes"; the other, where newcomers are developers who authored *not more than a median number of changes* is referred to as 'Median'.

3) Comparing the metrics: After splitting the observations into two sets by one of the strategies, we evaluated the difference in distributions of corresponding metrics in these sets. A difference in distributions of two subsets of a metric indicates the relationship between the 'newcomer' factor and the metric. The metrics selected for comparison are diverse and derived from the human interaction history. Thus, we were unable to make any assumptions about their exact distributions, but we assumed the distribution law to be the same for newcomers' and others' subsets of values of each metric. We tested the hypotheses of these two distributions being shift against each other – inequality of medians – with Mann-Whitney U test.

III. Results

1) RQ1: Reviewers' experience: Table I features the results of comparison of reviewers' experience metrics between changes authored by newcomers and by other developers for the studied systems.

The results vary between the systems. For most Eclipse projects the comparison reveals no significant difference in reviewers' experience. The possible reason is smaller project sizes and consequent smaller sizes of data samples, which require a stronger difference to be observed for the distributions to be considered significantly different. Selecting newcomers with median number of changes reveals a difference for more projects, which suggests that the newcomer effect in most cases takes place for some developers with more than 2 prior changes, which are not considered newcomers by the alternative selection method. When reviews count is used for reviewer expertise estimation, the direction of the difference is more stable across the whole system – where the difference is significant, newcomers' changes are reviewed by *less* experienced people. For the majority of the OpenStack projects, regardless of chosen experience metric and newcomer definition, the changes from newcomers are reviewed by *less* experienced developers. QT shows the most diverse results across the projects. The strongest effect is observed when past reviews count is used for reviewer experience estimation and newcomers are selected with a median – in 16 of the 20 projects newcomers' changes were reviewed by significantly *less* experienced people. Two of the other QT projects demonstrate very strong opposite effect in this case.

TABLE II: Comparison of review attention metrics

System	Metric	Newcomer definition	Metric is greater for newcomers ***	**	*	No sig. diff.	Metric is less for newcomers *	**	***
Eclipse	Comments count	Median	5	2	5	7	1	0	0
		$\leq$ 2 changes	1	1	1	**17**	0	0	0
	Reviewers count	Median	2	0	0	9	2	0	7
		$\leq$ 2 changes	0	0	0	**11**	4	3	2
	Time to first comment	Median	0	0	2	**14**	1	1	2
		$\leq$ 2 changes	0	0	1	**17**	2	0	0
	Review lifespan	Median	2	1	1	**15**	1	0	0
		$\leq$ 2 changes	0	0	0	**16**	3	1	0
OpenStack	Comments count	Median	3	1	0	4	1	0	**11**
		$\leq$ 2 changes	1	0	2	3	2	1	**11**
	Reviewers count	Median	6	0	0	7	0	3	4
		$\leq$ 2 changes	3	2	1	4	3	2	5
	Time to first comment	Median	0	1	1	1	3	1	**13**
		$\leq$ 2 changes	0	0	1	4	3	1	**11**
	Review lifespan	Median	6	2	0	8	0	1	3
		$\leq$ 2 changes	4	2	1	8	0	1	4
QT	Comments count	Median	7	2	1	4	4	0	2
		$\leq$ 2 changes	2	1	2	8	2	1	4
	Reviewers count	Median	6	4	0	5	1	0	4
		$\leq$ 2 changes	2	2	3	8	1	0	4
	Time to first comment	Median	1	0	0	**11**	5	1	2
		$\leq$ 2 changes	1	0	0	**14**	0	2	3
	Review lifespan	Median	5	0	2	**11**	0	1	1
		$\leq$ 2 changes	3	0	0	**15**	0	0	2

Numbers are counts of projects with corresponding effect and significance.
***: $p \leq .001$; **: $p \leq .01$; *: $p \leq .05$.

2) RQ2: Reviewers' attention: Table II presents the comparison of review attention metrics. Similarly to reviewers' experience, in most Eclipse projects the changes from newcomers do not significantly differ from other changes in terms of attention metrics. In most of the OpenStack projects, the number of comments and time to first comment are less in reviews of changes authored by newcomers. Number of reviewers and total review duration are different for newcomers' changes in most OpenStack projects, but the direction of the difference is not consistent. For most of QT projects, counts of comments and reviewers were less for newcomers' changes. However, a strong opposite result is observed for several projects. There is no significant difference for most of the QT projects in terms of the time-based metrics. Similarly to the comparison on the reviewers' experience, for all metrics a selection of newcomers using the median number of changes reveals the difference for more projects than selecting them by the maximum prior changes count of 2.

3) RQ3: Review outcomes: Results of review merge rate comparison are presented in Table III. Consistently across all three systems, we found newcomers' changes less likely to be eventually merged. Depending on newcomers selection method, changes authored by newcomers were 10 to 15% less likely to be merged than the other changes.

TABLE III: Comparison of review merge rate

	Newcomers defined by median			Newcomers defined by ≤ 2 changes		
System	Merge rate (newcomers)	Merge rate (others)	Ratio	Merge rate (newcomers)	Merge rate (others)	Ratio
Eclipse	0.75	0.87	0.86	0.74	0.86	0.86
OpenStack	0.76	0.88	0.86	0.74	0.87	0.85
QT	0.84	0.93	0.90	0.80	0.93	0.86

IV. Discussion

1) Findings: Our results indicate that at least in two of the three studied systems there exists an association between low author's experience with a project and code review process metrics. The newcomers' changes are reviewed differently in terms of *who* reviews their changes (reviewers' experience), *how* their changes are reviewed (attention metrics), and the likelihood of their changes to be merged.

The strength and direction of this difference vary across the systems. This suggests that the difference in actions towards newcomers is not a property of the code review process itself and is not imposed by the code review tool of their choice, but indicates some kind of special attitude to newcomers. This suggests that future research can be done to investigate more deeply the impact of status and other social factors on team's interaction, in terms of both identifying social phenomena in team's work artifacts and taking them into account in work towards the improvement of team collaboration tools.

Contrary to our expectations, we found reviewers of newcomers' changes to be more often *less* experienced in a project, than the other way around.

2) Threats to validity and future work: The metric that we used for empirical estimation of 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 remain the same through the project lifespan in general case. We illustrate this point with the following example: If a project that was initially developed by a group of people who continuously contribute to it, quickly gains popularity and attracts a lot of contributions from new people, the average number of contributions per developer decreases. This would lead to a significant increase of estimated experience for existing developers over a short period of time, which does not represent actual growth of their experience. However, we believe that this point does not affect the validity of our study, as we do not operate with the absolute values of this metric, but only use it to compare developers against each other. We plan to design an alternative method to empirically estimate a developer's project-wide experience in our future work.

Treating every project within the whole system independently, though providing a finer analysis granularity and revealing differences between the project in a system, does not let us consider a concept of a cross-project experience: the developers who typically contribute to related subsystems (which we considered here as different projects) should gain experience with a given subsystem faster thanks to their understanding of related parts and the whole system.

There is a similar effect related to cross-project experience, which makes our methods of newcomers' detection less precise. Our method considers as a newcomer a developer who contributed to other subsystems and had already gained solid experience and reputation, but is at the first contribution to a new subsystem. We also plan to address this issue in further work.

3) Comparison with existing work: Our results regarding reduced merge rate for newcomers is in line with existing results on this matter. Bosu *et al.* used social graph analysis to divide OSS contributors into core and peripheric groups based on *reputation* of developers for GitHub projects [22]. They found a significant difference between the groups in terms of patch acceptance rate and response time. Despite our experience model not taking the social network structure in a team into account directly, the similar results we obtained in this study confirm the capability of a straightforward count-based model to detect peripherical developers. Our work also demonstrates the presence of differences in newcomers' review characteristics for reviews performed with a different tool than GitHub. This point eliminates the workflow imposed by a code review tool as a possible source of observed inequality.

V. Conclusion

Understanding the impact of code review process on distribution of technical knowledge in the team, and vice versa, is vital for arming the code review tools with features to optimize the social outcomes of the process. In this study we focused on the association of author expertise with review process metrics. We explored the difference of some of the code review process metrics for changes authored by newcomers. We used the code reviews and changes history from the most actively developed parts of three large open source systems.

We found that in most cases the newcomers' changes are reviewed in a significantly different way. They differ in terms of reviewers' experience, number of comments, time before first comment, and other metrics. However, the strength and direction of the differences tend to vary across systems and projects.

This study suggests several directions for future work. The variation of effects between some projects and systems and their similarity among others suggests the existence of a collective mindset towards newcomers, the nature of which could be explored further.

References

[1] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 712–721.

[2] 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*, ser. ICSE. New York, NY, USA: ACM, 2014, pp. 345–355. [Online]. Available: /pub/exploration-pullreqs.pdf

[3] 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*. ACM, 2014, pp. 202–211.

[4] 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 *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 171–180.

[5] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 202–212.

[6] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75–90, 1992.

[7] 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 *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 141–150.

[8] M. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review," *IEEE Transactions on Software Engineering*, 2016.

[9] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead Jr, "Does bug prediction support human developers? findings from a google case study," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 372–381.

[10] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 31–41.

[11] C. W. Holsapple and A. B. Whinston, "Knowledge-based organizations," *The Information Society*, vol. 5, no. 2, pp. 77–90, 1987.

[12] K. Lewis, "Knowledge and performance in knowledge-worker teams: A longitudinal study of transactive memory systems," *Management science*, vol. 50, no. 11, pp. 1519–1533, 2004.

[13] 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.

[14] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the Fourth international Workshop on Computing Education Research*. ACM, 2008, pp. 3–14.

[15] 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.

[16] T. N. Bauer, T. Bodner, B. Erdogan, D. M. Truxillo, and J. S. Tucker, "Newcomer adjustment during organizational socialization: a meta-analytic review of antecedents, outcomes, and methods." *Journal of applied psychology*, vol. 92, no. 3, p. 707, 2007.

[17] G. R. Jones, "Socialization tactics, self-efficacy, and newcomers' adjustments to organizations," *Academy of Management journal*, vol. 29, no. 2, pp. 262–279, 1986.

[18] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 2003, pp. 419–429.

[19] 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.

[20] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 153–162.

[21] 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*. ACM, 2016, pp. 1039–1050.

[22] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 33.