Modern Code Reviews Process Properties

This briefing reports scientific evidence of 13 studies that observe the code review process and its properties.

FINDINGS

When code reviews should be performed?
Research shows that code reviews in large open source software projects are done frequently and in short intervals [PR2, PR5]. The results show that large and formal organizations might benefit from more frequent reviews and having invested reviewers (produced by overlap in developers' work) [PR5].
What we think: Evidence from large open-source projects shows that code reviews are done early and in short intervals. Code reviews should begin as soon as the change is submitted in order to capture the defects early.
What are the benefits of code reviews besides finding defects?
Analysis of large open source software projects found that the review acts as a group problem-solving activity where reviews support team discussions of defect solutions [PR2, PR5]. An analysis of over 100, 000 peer reviews found that reviews also enable developers and passive listeners to learn from the discussion [PR2, PR5].
What we think: There is evidence that code reviews are not only used to identify defects. It can be used as a collaborative tool where the overall knowledge of the code can be improved and joint discussions on solving problems can be done. This makes code reviews a good approach to support the onboarding of newcomers.
How review requests are distributed.
An analysis of five Apache projects found that broadcast reviews (e.g., mailing list) were twice faster. However, unicast reviews (e.g., Jira) were more effective in capturing defects [PR3]. In the same investigation, reviewers responded that unicast review allows them to comment on specific code, visualize changes and have less traffic of patches circulating among reviewers. However, new developers learn about the code structure faster with high traffic of patches circulated among those who subscribe to broadcast reviews.
What we think: Broadcast reviews are good to increase the overall knowledge sharing particularly for newcomers, while unicast facilitates the work of the code reviewers.
Efficiency and effectiveness of code reviews compared to team walkthroughs.
The code review process was compared with the walkthrough process in an industrial setting and was found to be very useful and efficient in comparison to walkthroughs [PR6].
What we think: Code reviews are lightweight and more effective compared to walkthroughs.
Mentioning peers in code review comments.
A study explored the current use of @mentions [PR7] and found that it is mostly used by code submitters. It reduces delay in developers' collaboration. Furthermore, it is more likely to be used in complex pull-requests that have more commits, comments, participants, and a longer time to handle.
What we think: There is evidence that @-mention can improve participation and response time. For critical issues, @-mention can help to get more attention to the review request.

Test code reviews.
Observations on code reviews found that the discussions on test code are related to testing practices, coverage, and assertions. However, test code is not discussed as much as production code [PR8]. When reviewing test code, developers face challenges such as lack of testing context, poor navigation support (between test and production code), unrealistic time constraints imposed by management, and poor knowledge of good reviewing and testing practices by novice developers [PR8].
What we think: Reviewing the test code to ensure good test coverage among other things will improve code maintainability and overall quality. Better tool support is needed to enable efficient and effective test code reviews, e.g., to navigate between test and production code.
Decision-making (integration, abandonment, resubmission) process in the code review process.
An analysis of the QT project showed that integrators only use the patch votes as a reference to make the decision to integrate or request a patch update [PR11]. However, patches that receive more negative votes are likely to be rejected.
What we think: Patch voting can be used when multiple reviewers review the same patch. It provides a quick overview of the review and also helps to look at decisions that were different from the majority vote. It can help in reducing bias.
Comparison of pre-commit and post-commit.
Pre-commit is commonly practiced in the form of pull requests and post-commits supports early and continuous integration whithatch may reduce conflicts. Comparison of pre-commit and post-commit reviews in 19 companies, ranging from startups to multinational companies found that there are no differences in most cases [PR13]. In some cases, post-commits were better regarding cycle time and quality. For pre-commit review, the review efficiency was better.
What we think: Evidence shows that there are no differences between pre and post-commit reviews. However, the performance of the reviews highly depends on the context such as the developer skills, number of developers, and conflict probability.
Strategies for merging pull requests.
A survey of developers and analysis of data from a commercial project found that pull request size, the number of people involved in the discussion of a pull request, author experience, and their affiliation are significant predictors of review time and merge decision [PR15]. It was found that developers determine the quality of a pull request by the quality of its description, its complexity, reversibility and the quality of the review process by the feedback quality, test quality, and the discussion among developers [PR15].
What we think: Evidence shows the factors that can be used to predict review time and merge decisions. Such predictions can be used to allocate resources and time for the review process. In addition, knowing the pull request qualities that reviewers look for will help in writing good patches.
Motivations, challenges and best practices of the code review process.
Analysis of the code review process at Microsoft found that improving code, finding defects and sharing knowledge were the top three out of nine identified benefits associated with code reviews [PR16].

Who is this briefing for?

Software engineering practitioners who want to make decisions about code reviews based on scientific evidence.

Where the findings come from?

All findings of this briefing were extracted from the research studies identified through a literature review.

What is included in this briefing?

Summaries of research papers in the form of takeaways for the practitioners.

To access other evidence briefings on code reviews:
https://rethought.se/modern-code-reviews/

For additional information about code review research contact -
deepika.badampudi@bth.se
michael.unterkalmsteiner@bth.se
ricardo.britto@bth.se

Similarly, Google found that knowledge sharing, history tracking, gatekeeping, and accident prevention as benefits of code reviews [PR18].
In Microsoft challenges such as receiving timely feedback, review size and managing time constraints were identified as the top three challenges out of 13 identified challenges [PR16 and PR17]. Whereas at Google, challenges such as geographical and organizational distance, misuse of tone and power, unclear review objectives/subject and context were identified as breakdowns of code review process [PR18].
The best practices for authors include writing small patches, describing and motivating changes, considering selecting reviewers and being receptive towards reviewers' feedback [PR16]. The reviewers should provide timely and constructive feedback through effective communication channels [PR16].
What we think: Evidence shows that code reviews have several benefits other than finding defects alone. Authors should consider sending good patches that are reviewable and reviewers should consider sending timely and constructive feedback.

References

ID	Title	Link
PR1	The Choice Of Code Review Process: A Survey On The State Of The Practice	http://tobias-baum.de/rp/stateofpractice.pdf
PR2	Convergent Contemporary Software Peer Review Practices	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.641.1046&rep=rep1&type=pdf
PR3	Broadcast Vs. Unicast Review Technology: Does It Matter?	http://swat.polymtl.ca/~foutsekh/docs/ICST-Tita.pdf
PR5	Contemporary Peer Review In Action: Lessons From Open Source Development	https://users.encs.concordia.ca/~pcr/paper/Rigby2012IEEE.pdf
PR6	Applying Continuous Code Reviews In Airport Operations Software	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
PR7	A Exploratory Study Of @-Mention In Github'S Pull-Requests	https://www.researchgate.net/profile/Yang_Zhang178/publication/283325670_A_Exploratory_Study_of_-Mention_in_GitHub%27s_Pull-Requests/links/5633816a08aeb786b7012f77/A-Exploratory-Study-of-Mention-in-GitHubs-Pull-Requests.pdf
PR8	When testing meets code review: Why and how developers review tests	https://dl.acm.org/doi/pdf/10.1145/3180155.3180192
PR11	Pilot Study Of Collective Decision-Making In The Code Review Process	http://library.naist.jp/dspace/bitstream/handle/10061/12706/111_CASCON2015.pdf?sequence=1&isAllowed=y
PR13	Comparing Pre-Commit Reviews And Post-Commit Reviews Using Process Simulation	https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1865?casa_token=TF0spbCXULYAAAAA%3A_WbHX1N6evg5u3pwdEQWViKLft9fWi6JuntjvRO_QVgFWBu3n5WljvQZHLpSmBQrN3jkOEW-2u9yJoc
PR15	Studying Pull Request Merges: A Case Study Of Shopify'S Active Merchant	https://dl.acm.org/doi/pdf/10.1145/3183519.3183542
PR16	Code Reviewing In The Trenches: Challenges And Best Practices	https://pdfs.semanticscholar.org/2ba0/85afac89d6d5491bff9e93d6658a945cca10.pdf
PR17	Expectations, Outcomes, And Challenges Of Modern Code Review	https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICSE202013-codereview.pdf
PR18	Modern Code Review: A Case Study At Google	https://dl.acm.org/doi/pdf/10.1145/3183519.3183525