The Impact of Software Development Processes, patch Characteristics, and Tools on Modern Code Reviews

This briefing reports scientific evidence of 11 studies that investigate the impact of software development processes, tool support, and patch characteristics on code reviews.

FINDINGS

The impact of static analyzers on the code review process.
An analysis of six open source projects found that static analyzers such as PMD and ChangeStyle can be used to identify and consider warnings related to imports, regular expression, and type resolution in code reviews [ICR1].
Through an experiment [ICR3], it was found that the use of a symbolic execution debugger to identify defects during the code review process is effective and efficient compared to a plain code-base view.
What we think: Static code analyzers and formal methods such as symbolic execution debuggers can be used to support the code review process. Configuring the static analyzers to the project's requirements will make analyzers effective. Although the evidence on symbolic execution effectiveness is derived from a small experiment and more evidence is needed to support this claim.

The impact of gamification elements on the code review process.
An experiment with gamification elements in the code review process found that there is no impact of gamification on the identification of defects [ICR2].
What we think: Use of gamification does not seem to improve the identification of defects. However, more investigation is needed to fully evaluate its impact on different contexts.
The impact of continuous integration on the code review process.
Experiments with 26,516 automated build entries reported that successfully passed builds are more likely to improve code review participation and frequent builds are likely to improve the overall quality of the code reviews [ICR6].
What we think: Build logs could be used to identify the pull requests for code review.
The impact of code change descriptions on the code review process.
Interviews with industrial and OSS developers conclude that providing motivations for code changes along with a description of what is changed reduces the reviewer burden [ICR5]. Similarly, an analysis of Android, Qt, and OpenStack projects found that a short patch description can lower the likelihood of attracting reviewers [ICR4].
What we think: Feedback on the change descriptions can be provided to authors either manually or through an automated tool which might result in good code changes. Good code changes help in attracting reviewers.
The impact of code size changes on the code review process.
An investigation of a large commercial project with 269 repositories found that when patch size increases, the reviewers become less engaged and provide less feedback [PR12].
An interview study with industrial and open source software project developers found that code changes that are properly sized are more reviewable [ICR5].
The size of patches negatively affects the review response time, as observed in a study on Webkit's and Blink's code reviews [HF11], and reduces the number of review comments, as shown in a study of the Chromium project [OG3].
Similarly, an analysis of more than 100,000 peer reviews of the projects httpd server, Subversion, Linux, FreeBSD, KDE, and Gnome recommends that changes to be reviewed should be small, independent, and complete [PR5].
What we think: Code churn impacts reviewer engagement and participation. Therefore, it is important to ensure that the submitted code is not too large.
The impact of commit history coherence on the code review process.
An interview study on industrial and open-source software project developers found that the commit messages that are self-explanatory and have meaningful messages are easier to review [ICR5]. In addition, interviewees suggest that the ratio of commits in a change to the number of files changed should not be high [ICR5].
What we think: Commit history should be meaningful and should help in understanding the change logic which will help in reviewing the change. Too many commits with minor fixes should be avoided and should be limited to local branches.
The impact of review participation history on the code review process.
An analysis of Android, Qt, and OpenStack open-source software projects found that the likelihood of attracting reviewers is higher when past changes to the modified files are reviewed by at least two reviewers [ICR4]. Prior patches that had few reviewers tend to be ignored [ICR4].
Another study, looking at reviews from the Webkit and Blink open source projects, found that more active reviewers have faster response times [HF11].
What we think: Ensuring good review coverage will help in attracting reviewers both in industrial and open-source software projects. Furthermore, in case of time constraints, reviewers that were active in the past tend to deliver timely reviews.
The impact of fairness on the code review process.
Fairness, in general, refers to the decision and allocation of resources in a way that is fair to the individuals and the group. A study [OG14] in the Openstack project investigated different fairness aspects and recommends, besides the common aspects of fairness such as politeness, and precise and constructive feedback, to: (a) distribute reviews fairly, and (b) establish a clear procedure for how reviews are performed.
What we think: Since code reviews consist also of social interactions, it is important to create a climate in which participants have a positive experience, that fosters performance and collaboration. Relatively little has been studied in this area and it would be interesting how fairness is addressed in different contexts.

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

References

ID	Title	Link
ICR1	Would Static Analysis Tools Help Developers With Code Reviews?	https://www.academia.edu/download/48461162/Would_Static_Analysis_Tools_Help_Develop20160831-1998-1ytbjg0.pdf
ICR2	Impact Of Gamification On Code Review Process - An Experimental Study	https://dl.acm.org/doi/pdf/10.1145/3021460.3021474
ICR3	Can Formal Methods Improve The Efficiency Of Code Reviews?	http://envisage-project.eu/wp-content/uploads/2013/09/CodeReviewEval.pdf
ICR4	Review Participation In Modern Code Review: An Empirical Study Of The Android, Qt, And Openstack Projects	https://pdfs.semanticscholar.org/e65c/c76dc3173658abb3c77dc5010546c6b79238.pdf
ICR5	What Makes A Code Change Easier To Review: An Empirical Investigation On Code Change Reviewability	https://dl.acm.org/doi/pdf/10.1145/3236024.3236080
ICR6	Impact Of Continuous Integration On Code Reviews	https://arxiv.org/pdf/1807.01851.pdf
PR5	Investigating The Effectiveness Of Peer Code Review In Distributed Software Development	https://users.encs.concordia.ca/~pcr/paper/Rigby2012IEEE.pdf
PR12	An Empirical Study Of The Impact Of Modern Code Review Practices On Software Quality	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
HF11	Investigating Technical And Non-Technical Factors Influencing Modern Code Review	http://svn-plg.uwaterloo.ca/~migod/papers/2015/ese_2015_baysal.pdf
OG3	Analyzing Involvements Of Reviewers Through Mining A Code Review Repository	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
OG11	Let'S Make It Fun: Gamifying And Formalizing Code Review	https://www.scitepress.org/papers/2016/59372/pdf/index.html