Modern Code Review Performance and Human Factors

This briefing reports scientific evidence of 18 studies that investigate human factors and their relationship to modern code review performance.

FINDINGS

Review performance and reviewers' age and experience.

A study on over 100,000 peer reviews from open source projects (Apache, Subversion, Linux Kernel, FreeBSD, KDE and Gnome) found that sub-system expertise is a good indicator for review quality [PR5]. This relationship has also been observed in Microsoft [PR14], Mozilla [OG12, HF10], Qt, and Openstack [OG16].

Files reviewed by experienced reviewers showed also to be less vulnerable, as a study on the Chromium browser showed [OG8].

A survey at Mozilla also found that reviewer experience is an important factor for review time and if a patch is accepted or rejected [HF10].

A study on many open source projects (Android, LibreOffice, Openstack, QT) showed that the likelihood of a developer accepting a review task depends to a large degree on his or her familiarity with the code [HF1]. If only one expert exists on a sub-system, one strategy could be to always select the same reviewer to establish expertise over time. Another suggestion is to let senior developers self-select the code they are interested in and are competent to review.

A study on reviews in the Eclipse, Openstack and QT projects investigated the influence of experience (both of developer and reviewer) on code reviews [HF6]. The findings suggest that contributions by new developers are not reviewed by experienced reviewers. Also, code from new developers does not receive more attention during review. However, code from new developers is less likely to be merged.

A similar observation was made in a study on reviews in the Webkit and Blink open source projects [HF11]. Another study investigated if age affects reviewing performance [HF8]. The study compared students in their 20's and 40's showed no difference based on age or development experience.

Finally, there exists some early work on harvesting reviewer experience through crowdsourcing the creation of rules and suggestions [HF12].

What we think: There is clear evidence that reviewer experience has an impact on review quality. This is not surprising in itself, might however be helpful when developing systems that suggest reviewers automatically. These systems could also make sure that reviewers get a good mix of reviews on familiar code and unknown code so that they expand their expertise over time. Experienced reviewers possess tacit knowledge that is difficult to formalize and convey to novice programmers. Systems that could mine this knowledge from reviews would be an interesting avenue for research.

Review performance and reviewers' reviewing patterns and focus.

Eye tracking has been used in several studies to investigate how developers review code. Researchers found that a particular eye movement, the scan pattern, is correlated with defect detection speed [HF2, HF3, HF5]. In the scan pattern, the reviewer first reads briefly the code from top to bottom and then focuses on particular portions. The more time the developer spends on scanning, the more efficient is the defect detection [HF3]. Based on these results, researchers have also stipulated that reviewing skill and defect detection capability can be deduced from eye movement [HR7].

What we think: Eye tracking is an interesting approach to study how code reviewers perform their task. It would be interesting to study if IDEs or code review tools could be adapted to support the scanning pattern. Until such support exists, a sensible recommendation would be that novice reviewers should first read the whole code, to get an overview, and then focus on individual parts.

Review performance and reviewers' workload.

The impact of workload on code reviews has been investigated from two perspectives. First, a study on the Mozilla project found that workload (measured in pending review requests) negatively impacts review quality in terms of bug detection effectiveness [OG12]. Second, a study crossing several open source projects (Android, LibreOffice, Openstack, QT) found that workload (measured in concurrent and remaining review tasks) negatively impacts the likelihood that the reviewers accepts a new review invitation [HF1].

What we think: While both findings are not surprising, the studies provide some compelling evidence that reviewer workload must be considered when distributing reviewing tasks. Since these studies were conducted in open source projects, it would be interesting if the findings hold true also in closed source software development.

Review performance and reviewers' social interactions.

Code reviews have been studied with different theoretical lenses on social interactions. A study on the Android, Openstack and QT open source projects used social network analysis to model reviewer relationships and found that the most active reviewers are at the center of peer review networks [OG2].

Another study, again looking at reviews from Openstack and QT, used the snowdrift game (a game similar to the famous prisoners dilemma) to model the motivations of developers participating in code reviews [OG6]. They describe two motivations: (i) a reviewer has a motive of choosing a different action (review, not review) from the other reviewer, and (ii) a reviewer cooperates with other reviewers when the benefit of review is higher than the cost.

Finally, a study on the Mozilla project found that past participation in reviews on a particular subsystem is a good predictor for accepting future review invitations [HF1]. Other factors, such as code authoring experience, and familiarity between the reviewer and the patch author, also play a role in the decision of accepting review invitations.

What we think: Using game and social network theory on the peer relationships of code reviewers and developers is an interesting angle. We are curious how these results can be used to support practical decision making, e.g., when planning for expertise redundancy or matching reviewers.

Review performance and reviewers' understanding of each other's code.

A study on the reviews in the Android project investigated if reviewers' confusion can be detected by humans and if a classifier can be trained to detect reviewers' confusion in review comments [HF9]. The study defines confusion in seven categories: hedges, probables and hypotheticals representing indirect expressions of uncertainty; questions requesting a solution; I statements and nonverbals describing direct psychological expression of uncertainty; and meta capturing the discussion of uncertainty. While humans are quite capable of detecting confusion, automated detection is still challenging.

What we think: Knowing when code and code reviews could lead to misunderstandings is a powerful decision aid that can be used to train developers and reviewers. Recognizing confusion in reviews could also be useful to direct the attention of experienced reviewers that could help to clarify the situation.

Review performance and reviewers' perception of code and review quality.

A survey study conducted among reviewers in the Mozilla project identified factors that determine their perceived quality of code and code reviews [HF10]. High quality reviews provide clear and thorough feedback, in a timely manner, by a peer with a supreme knowledge of the code base, strong personal and interpersonal qualities. Challenges to achieve high quality reviews are of technical (familiarity with the code, coping with code complexity, suitable tool support) and personal (time management, technical skills and context switching) nature.

What we think: This study provides a snapshot of a particular context (an open source project). It would be interesting to study whether and how tradeoffs among code and review quality aspects are handled differently in other 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
HF1	The Impact Of Human Factors On The Participation Decision Of Reviewers In Modern Code Review	http://socsel.sys.wakayama-u.ac.jp/publications/journal/EMSE2018_Shade.pdf
HF2	Exploiting Eye Movements For Evaluating Reviewer'S Performance In Software Review	https://uwanolab.jp/pman/data/pdf/7.pdf http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.8125&rep=rep1&type=pdf
HF3	An Eye-Tracking Study On The Role Of Scan Time In Finding Source Code Defects	http://www.cs.kent.edu/~jmaletic/papers/ETRA12.pdf http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.381.2034&rep=rep1&type=pdf
HF5	Eye Movements In Code Review	https://andrewbegel.com/papers/eye-movements-code-review.pdf
HF6	Code Review for Newcomers: Is It Different?	https://sback.it/publications/chase2018.pdf
HF7	A Fuzzy Inference System To Recommend Skills For Source Code Review Using Eye Movement Data	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
HF8	Wap: Does Reviewer Age Affect Code Review Performance?	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 http://129.97.186.80/~migod/papers/2015/ese_2015_baysal.pdf
HF10	Code Review Quality: How Developers See It	https://plg.uwaterloo.ca/~migod/papers/2016/icse16.pdf http://olgabaysal.com/pdf/icse2016.pdf
HF12	Towards Crowdsourced Large-Scale Feedback For Novice Programmers	http://michin01.github.io/vlhcc2014gc.pdf
HF9	Confusion Detection In Code Reviews	http://www.win.tue.nl/~aserebre/ICSME2017Felipe.pdf
OG2	Social Network Analysis In Open Source Software Peer Review	https://www.researchgate.net/profile/Xin_Yang73/publication/283442347_Social_Network_Analysis_in_Open_Source_Software_Peer_Review/links/5638654108ae4bde50213382.pdf
OG6	Code Review Participation: Game Theoretical Modeling of Reviewers in Gerrit Datasets	https://www.researchgate.net/profile/Hideaki_Hata/publication/303323826_Code_review_participation_game_theoretical_modeling_of_reviewers_in_gerrit_datasets/links/5b3c4725a6fdcc8506eee153/Code-review-participation-game-theoretical-modeling-of-reviewers-in-gerrit-datasets.pdf
OG8	An Empirical Investigation Of Socio-Technical Code Review Metrics And Security Vulnerabilities	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
OG12	Investigating Code Review Quality: Do People And Participation Matter?	http://svn-plg.uwaterloo.ca/~migod/papers/2015/icsme15-OleksiiOlgaLatifa.pdf
OG16	Revisiting Code Ownership And Its Relationship With Software Quality In The Scope Of Modern Code Review	https://sail.cs.queensu.ca/Downloads/ICSE2016_RevisitingCodeOwnershipAndItsRelationshipWithSoftwareQualityInTheScopeOfModernCode%20Review.pdf
PR5	Contemporary Peer Review In Action: Lessons From Open Source Development	https://users.encs.concordia.ca/~pcr/paper/Rigby2012IEEE.pdf
PR14	Code Reviews Do Not Find Bugs. How The Current Code Review Best Practice Slows Us Down	https://www.microsoft.com/en-us/research/wp-content/uploads/2015/05/PID3556473.pdf