SUPPORT SYSTEMS FOR MODERN CODE REVIEWS (III)

This briefing reports scientific evidence of 28 studies that investigate support systems for the recommendation of reviewers, automation of reviews and code reviews on touch enabled devices.

FINDINGS

Reviewer recommendations.

The recommendation of adequate reviewers in modern code reviews is a very popular topic in the related research community. Most papers related to this theme focus on proposing tools to recommend reviewers and validate their approaches using historical data extracted from open source projects.

Most approaches (15 approaches) recommend code reviewers based on the similarity between files modified or reviewed by each developer and the files of a new pull request [RR1, RR3, RR5, RR7, RR8, RR11-RR17, RR20-RR22]. While in most cases all files included in a pull request are considered, one approach only considers external files mentioned in a pull request [RR20]. In some cases, file path similarity is the sole predictor used to recommend reviewers [RR1, RR7, RR14, RR17, RR20, RR21], while in others it is combined with other predictors such as previous interactions between submitter and potential reviewers [RR3, RR11, RR12, RR16], pull request content similarity [RR3, RR8, RR12], contribution to similar files [RR5, RR13], developer activeness in a project [RR3, RR5, RR11, RR15] and file metadata [RR22].

Another popular predictor used to recommend code reviewers (7 approaches) is the similarity between the content of previous and new pull requests [RR3, RR4, RR6, RR8, RR9, RR10, RR12]. This similarity is measured using topic modelling. Half of the identified recommendation approaches rely only on this predictor [RR4, RR9, RR10], while the others either combine it with the file path similarity (as shown above), or previous interactions between submitter and potential reviewers.

Out of all approaches identified by us, only one does not include file path similarity or pull request similarity [RR2]. Instead, the authors combined the metadata of pull requests (e.g., number of files) with the metadata associated with potential reviewers (e.g., contributor type and number of overall contributions). In relation to how the predictors are used to recommend code reviewers, most approaches employ traditional approaches (e.g., cosine similarity), while a few use machine learning technics, such as Random Forest [RR2] Naive Bayes [RR2], Context-Aware Collaborative Filtering [RR22], and Support Vector Machines [RR6, RR11]. One approaches code reviewer recommendation as an optimization problem and uses Genetic Algorithms.

The performance of the identified approaches varies a lot and is often measured using Accuracy [RR2, RR4, RR7, RR11, RR13, RR20, RR22], Precision and Recall [RR1, RR3, RR6, RR9, RR12, RR16, RR20, RR21], or Mean Reciprocal Rank [RR1, RR8, RR11, RR21, RR22]. Considering that it is not possible to compare the values reported in all analyzed studies due to different validation setups, overall the results show that there is still room to improve the performance of existing recommenders.

Another aspect that is worth mentioning is that out of the 22 studies we analyzed, only three [RR5, RR17, RR22] have evaluated code reviewer recommendation tools in live environments, while just one has [RR22] has checked the accuracy based on the opinion of users and the opinion of the recommend developers. Instead, the majority of the studies measures performance (accuracy, precision, recall, and mean reciprocal rank) by comparing the actual list of reviewers present in historical data with the list of developers recommended by their respective approaches. This type of approach to evaluate recommendation performance misses the fact that a developer recommended by tool might be adequate but was not in the list of actual reviewers, which leads to underestimation of the performance associated with most approaches analyzed by us.

Finally, one study focuses on identifying factors that should be accounted for when recommending reviewers [RR18]. They have identified that factors such as the number of files and commits in a pull request, pull requester profile, previous interactions between contributors, previous experience with related code, and ownership of modified code are factors related to how code reviewers are selected.

What we think: There is little evidence of the feasibility of code reviewer recommenders in production environments. This is due to the fact that most studies have used historical data to validate their respective approaches. Furthermore, the historical data used in the existing studies mostly comes from open source projects, which is an additional limitation of the state of the art. We believe more research is necessary to show the feasibility of code reviewer recommenders in real environments. Research must focus on other contexts as well in addition to open source projects.

Automating code reviews.

Researchers have proposed to train a classifier on whether a change request is likely to be accepted or not [RP6]. Knowing in advance the likelihood of a rejected change request would reduce the review effort as those changes would not even reach the reviewing stage. The researchers use the following historic information as input for their classifier: the change request owner, the assigned reviewers, project context, change summaries, review results, and revision counts. They found that the change requests by inexperienced developers that involve many reviewers are the most likely to be rejected.

Review Bot, an extension to the code review tool Review Board, uses multiple static code analysis tools (PMD, Checkstyle) to check for common defect patterns and coding standard violations to create automated code reviews [SRA4]. An evaluation with seven developers found that they agreed to 93% of the automatically generated comments, likely due to the lack of consistent adoption of coding standards, which were the majority of the identified defects.

In a similar way, researchers studied the overlap of PMD findings with reviewer comments in 92 pull requests from GitHub [SRA3]. Of 274 comments, 43 overlapped with PMD warnings, indicating that 16% of the review workload could have been reduced with automated review feedback.

CFar, developed at Microsoft, has been used in a production environment resulting in: (a) enhanced team collaboration as analysis comments were discussed; (b) improved productivity as the tool freed developers from providing feedback about shallow bugs; (c) improved code quality since the flagged issues were acted upon; and (d) the automatic review comments were found useful by the 98 participating developers [SRA2].

What we think: Reducing the review effort by providing automated review feedback is certainly interesting as it allows for administering these tools centrally (not relying on individual developers installing them), keep rules up-to-date, run the analysis, and act on the results. However, as with all static code analyzers, it is important to keep false positives under control, otherwise the reports will either be ignored or lead to additional unnecessary review effort.

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

Code reviews on touch enabled devices.

Researchers have proposed to use multi-touch devices, such as the Microsoft Surface Table, for collaborative code reviews, in an attempt to make the review process more desirable [D17]. The approach provides visualizations, for example to illustrate code smells (indicators for bad code) and metrics.

Other researchers have compared reviews performed on the desktop and on mobile devices (the mobile application is a specifically developed front-end for Gerrit) [D8]. In an experiment, they analyzed 2,500 comments, produced by computer science students and found that: (a) the reviewers on the mobile device found as many defects as the ones on the desktop; and (b) seemed to pay more attention to details (speculatively, because only a few lines of code could be shown at once).

What we think: The main question is, why would touch-enabled devices provide benefits for the reviewing process. A table-top computer with touch interface could replace virtual meetings and the large screen would allow multiple reviewers to review the code at the same time. But again, what is the benefit there? On mobile devices, the benefits are even less obvious as the small screen limits the reviewer in gaining an overview on large change sets.

References

ID	Title	Link
D8	Experimental Validation Of Source Code Reviews On Mobile Devices	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
D17	An Approach For Collaborative Code Reviews Using Multi-Touch Technology	https://www.zora.uzh.ch/62916/1/20120604161811_merlin-id_7014.pdf
SRA2	Cfar: A Tool To Increase Communication, Productivity, And Review Quality In Collaborative Code Review	https://mariachris.github.io/Pubs/CHI-2018.pdf
SRA3	Evaluating How Static Analysis Tools Can Reduce Code Review Effort	https://people.umass.edu/bijohnson/docs/vlhcc2017.pdf
SRA4	Reducing Human Effort And Improving Quality In Peer Code Reviews Using Automatic Static Analysis And Reviewer Recommendation	https://www.researchgate.net/profile/Vipin_Balachandran2/publication/261501912_Reducing_human_effort_and_improving_quality_in_peer_code_reviews_using_automatic_static_analysis_and_reviewer_recommendation/links/5ce4439d458515712eba57aa/Reducing-human-effort-and-improving-quality-in-peer-code-reviews-using-automatic-static-analysis-and-reviewer-recommendation.pdf
RP6	Will It Pass? Predicting The Outcome Of A Source Code Review	http://online.journals.tubitak.gov.tr/openAcceptedDocument.htm?fileID=887608&no=212187
RR1	Profile Based Recommendation Of Code Reviewers	https://link.springer.com/content/pdf/10.1007/s10844-017-0484-1.pdf
RR2	Developers Assignment For Analyzing Pull Requests	https://dl.acm.org/doi/pdf/10.1145/2695664.2695884
RR3	Who Should Comment On This Pull Request? Analyzing Attributes For More Accurate Commenter Recommendation In Pull-Based Development	Link
RR4	An Empirical Study Of Reviewer Recommendation In Pull-Based Development Model	https://dl.acm.org/doi/pdf/10.1145/3131704.3131718
RR5	Does Reviewer Recommendation Help Developers?	https://pure.tudelft.nl/portal/files/46774016/revrec_preprint.pdf
RR6	Reviewer Recommender Of Pull-Requests In Github	https://www.trustie.net/attachments/download/84261/Reviewer%20Recommender%20of%20Pull-Requests%20in%20GitHub.pdf
RR7	Improving Code Review Effectiveness Through Reviewer Recommendations	https://dl.acm.org/doi/pdf/10.1145/2593702.2593705
RR8	Who Should Review This Change?: Putting Text And File Location Analyses Together For More Accurate Recommendations	https://xin-xia.github.io/publication/icsme15.pdf
RR9	Topic-Based Integrator Matching For Pull Request	https://arxiv.org/pdf/1710.10421.pdf
RR10	Understanding Review Expertise Of Developers: A Reviewer Recommendation Approach Based On Latent Dirichlet Allocation	https://www.mdpi.com/2073-8994/10/4/114/pdf
RR11	Coredevrec: Automatic Core Member Recommendation For Contribution Evaluation	https://link.springer.com/content/pdf/10.1007/s11390-015-1577-3.pdf
RR12	Earec: Leveraging Expertise And Authority For Pull-Request Reviewer Recommendation In Github	https://dl.acm.org/doi/pdf/10.1145/2897659.2897660
RR13	Automatically Recommending Code Reviewers Based On Their Expertise: An Empirical Comparison	https://dl.acm.org/doi/pdf/10.1145/2970276.2970306
RR14	Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach For Modern Code Review	https://library.naist.jp/dspace/bitstream/handle/10061/12737/10061_12737.pdf?sequence=1
RR15	Revrec: A Two-Layer Reviewer Recommendation Algorithm In Pull-Based Development Model	https://link.springer.com/content/pdf/10.1007/s11771-018-3812-x.pdf
RR16	Search-Based Peer Reviewers Recommendation In Modern Code Review	https://ouniali.github.io/papers/ICSME2016.pdf
RR17	Exploring How Software Developers Work With Mention Bot In Github	https://link.springer.com/content/pdf/10.1007/s42486-019-00013-2.pdf
RR18	What Factors Influence The Reviewer Assignment To Pull Requests?	Link
RR19	A Code Reviewer Assignment Model Incorporating The Competence Differences And Participant Preferences	https://content.sciendo.com/downloadpdf/journals/fcds/41/1/article-p77.xml
RR20	Correct: Code Reviewer Recommendation In Github Based On Cross-Project And Technology Experience	https://dl.acm.org/doi/pdf/10.1145/2889160.2889244
RR21	Automatically Recommending Peer Reviewers In Modern Code Review	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
RR22	Using a Context-Aware Approach to Recommend Code Reviewers: Findings from an Industrial Case Study	Please contact one of the authors of this evidence briefing to receive a copy of this paper.