Support Systems for Modern Code Reviews (I)

This briefing reports scientific evidence of 23 studies that investigate support systems related to the usefulness, sentiment, order, and monitoring of modern code reviews.

FINDINGS

Determining the usefulness of code reviews.
A study of three Github hosted projects, developed a taxonomy of review comments [RC1, RC3]. The categories are comments regarding code correctness (style checking, defect detecting, code testing), pull request decision making (value affirming, solution disagreeing, further questioning), project management (road-map managing, reviewer assigning, convention checking), and social interaction (politely responding, contribution encouraging). The researchers trained a classifier and categorized 147K comments, finding that inexperienced contributors tend to produce code that passes tests while still containing issues, and external contributors break project conventions in their early contributions.
In another study, researchers analyzed the usefulness of 1,116 review comments (a manual process that has also been attempted to automatize [RC9]) in a commercial system [RC8]. They marked a comment as useful if it triggered a code change within its vicinity (up to 10 lines) and analyzed features of the review comment pertaining to its content and author. The results indicate that useful comments share more vocabulary with the changed code, contain relevant code elements, and are written by more experienced reviewers.
A study of 2,817 review comments from the Openstack project found that only about 14% of comments are related to software design [RC2]; of which 73% provided suggestions to address the concerns, indicating that they were useful.
A study at Microsoft investigated the characteristics of useful code reviews by interviewing seven developers [RC6]. The study found that the most "useful" comments identify functional issues, scenarios where the reviewed code fails, and suggest API usage, design patterns or coding conventions. "Useless" comments ask how an implementation works, praise code, or point to work needed in the future. Armed with this knowledge, the researchers trained a classifier that achieved 86% precision and 93% recall in identifying useful comments. Applying the classifier on 1.5M review comments, they found that: (a) reviewer experience with the code base correlates with usefulness of comments, suggesting that reviewer selection is crucial, (b) the smaller the changeset, the more useful the comments, and (c) a comment usefulness density metric can be used to pinpoint areas where code reviews are ineffective (e.g. configuration and build files).
Criticism of the above pure statistical, syntactic approaches arose as the actual meaning of comments is not analyzed [RC7]. Linguists could help to provide guidelines for good writing practices, develop tools such that these good writing practices are enforced, and help to identify rationales in review comments that could be collected for documentation purposes.
What we think: The conducted research to eliminate vagueness and determine characteristics of usefulness is quite useful as it helps to determine objective criteria that can be operationalized to check usefulness of comments automatically. This opens the door to support systems that could provide feedback while writing a comment or allow the analysis of large data sets to understand the consequences of good and bad review comments.

Analyzing sentiments, attitudes and intentions in code reviews.
Understanding review comments in greater detail could lead to systems that support reviewers in both formulating and interpreting the intentions of code reviews. A study on Android code reviews investigated the communicative goals of questions stated in reviews [RC10], identifying five different intentions: suggestions (where action requests are formulated as questions), requests for information, attitudes and emotion (like criticism, anger, surprise), description of a hypothetical scenario, and rhetorical questions. The study suggests that the purpose of questions in reviews is not solely to obtain information.
Another study looked in particular at the sentiments expressed in reviews [RC5] and developed a classifier that flags comments as positive, neutral or negative with 83% accuracy. Finally, a study on the Chromium project found that code reviews with lower inquisitiveness, higher sentiment (positive or negative) and lower syntactic complexity were more likely to miss vulnerabilities in the code [RC4].
What we think: If we consider comments in code reviews as means to convey knowledge, it is important to study how different forms of expression support or detract from effective communication. People communicate in subtle ways, using sarcasm and irony that are notoriously hard to detect for automated systems.
Optimizing the order of reviews.
Code reviewers often need to prioritize which changes they should focus on reviewing first. Researchers propose to base the review decision on the likelihood that a particular change will eventually be merged [RP2]. They suggest using 34 features, grouped into five dimensions (code, file history, owner experience, collaboration network, and commit text) and train a random forest machine learning model.
The evaluation on three open source projects suggests that their approach is better than a random guess. PRioritizer is a pull request prioritization approach that, similar to a priority inbox, sorts requests that require immediate attention to the top [RP3]. The decision on priority is based on historical data to predict the likelihood that a pull request will be updated in the near future. Users of the system reported that they liked the prioritization, miss however insights on the rationale for the particular pull requests.
Other studies looked especially at the historic proneness of files to defects to direct review efforts. A study suggests combining bug-proneness, estimated review cost, and the state of a file (newly developed or changed) to prioritize files to review [RP4]. The evaluation, performed on two open source projects, indicates that the approach leads to more effective reviews.
A similar approach attempts to classify files as potentially defective, based on historic instances of detected faults and features on code ownership, and change request and source code metrics [RP5].
What we think: In scenarios where dedicated code reviewers, so called integrators, need to decide what to review next, the proposed approaches might provide some support. Nevertheless, there exists still very little evidence on the effectiveness of the suggested prioritization techniques in industry-grade environments as the evaluations have been performed in simulations or in the context of small user studies.
Monitoring review performance and quality. Researchers have proposed to quantitatively study the code reviews process, based on traces left in software repositories [RA2]. Without having access to any code review tool, they analyzed changelog files, commit records, and attachments and flags in Bugzilla records to monitor the size of the review process (actors involved and actions performed), the effort involved (number of iterations), and process delay (delay between request and review).
A similar study on the Xen and Netdev projects focused on review messages wherein changes are first reviewed through communication is a mailing list [RA4]. They developed a series of metrics to characterize code review activity, effort, and delays [RA1], which are also provided through a dashboard that shows the evolution of the review process over time [RA3].
Another study looked at code reviews managed with Gerrit and proposed metrics to measure velocity and quality of reviews [RA5].
Similar metrics, such as code churn, modified source code files and program complexity, were used to analyze reviewer effort and contribution in the Android open source project [RA7].
Other tools to analyze Gerrit review data are ReDa, that provides reviewer, activity and contributor statistics [D21], and Bicho, that models code reviews as information from an issue tracking system, allowing to query review statistics with standard SQL queries [D23]. Finally, Codeflow Analytics aggregates and synthesizes code review metrics (over 200) and was developed and evaluated at Microsoft [D22]. The tool provides access to analysis reports via Excel templates but also through more powerful SQL queries.
What we think: Monitoring and analyzing the review process over time can be used to improve the review process, e.g., by providing training to inexperienced reviewers and finding bottlenecks in tooling or overworked reviewers.

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
RC1	Automatic Classification Of Review Comments In Pull-Based Development Model	https://whystar.github.io/res/paper/seke2017.pdf
RC2	An Empirical Study Of Design Discussions In Code Review	http://rebels.ece.mcgill.ca/papers/esem2018_elzanaty.pdf
RC3	What Are They Talking About? Analyzing Code Reviews In Pull-Based Development Model	https://whystar.github.io/res/paper/jcst2017.pdf
RC4	Natural Language Insights From Code Reviews That Missed A Vulnerability: A Large Scale Study Of Chromium	https://nuthanmunaiah.github.io/static/assets/natural-language-insights.pdf
RC5	Senticr: A Customized Sentiment Analysis Tool For Code Review Interactions	http://amiangshu.com/papers/senticr-ase.pdf
RC6	Characteristics Of Useful Code Reviews: An Empirical Study At Microsoft	https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bosu2015useful.pdf
RC7	Code Review Comments: Language Matters	https://arxiv.org/pdf/1803.02205
RC8	Predicting Usefulness Of Code Review Comments Using Textual Features And Developer Experience	https://arxiv.org/pdf/1807.04485
RC9	Assessing Mcr Discussion Usefulness Using Semantic Similarity	https://www.researchgate.net/profile/Patanamon_Thongtanunam/publication/289429518_Assessing_MCR_Discussion_Usefulness_Using_Semantic_Similarity/links/57be6d8908aeda1ec3862ca7/Assessing-MCR-Discussion-Usefulness-Using-Semantic-Similarity.pdf
RC10	Communicative Intention In Code Review Questions	https://felipeebert.github.io/publications/icsme2018.pdf
RA1	Using Metrics To Track Code Review Performance	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
RA2	Code Review Analytics: Webkit As Case Study	https://www.researchgate.net/profile/Gregorio_Robles/publication/289147353_Code_Review_Analytics_WebKit_as_Case_Study/
RA3	Software Development Analytics For Xen: Why And How	https://www.researchgate.net/publication/326337424_Software_Development_Analytics_for_Xen_Why_and_How
RA4	Characterization of the Xen Project Code Review Process: An Experience Report	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
RA5	Metrics For Gerrit Code Reviews	http://ceur-ws.org/Vol-1525/paper-03.pdf
RA7	Mining Peer Code Review System For Computing Effort And Contribution Metrics For Patch Reviewers	Please contact one of the authors of this evidence briefing to receive a copy of this paper.
D21	Reda: A Web-Based Visualization Tool For Analyzing Modern Code Review Dataset	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.7617&rep=rep1&type=pdf
D22	Lessons Learned From Building And Deploying A Code Review Analytics Platform	http://cabird.com/pubs/bird2015cfa.pdf
D23	Analyzing Gerrit Code Review Parameters With Bicho	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.672.8656&rep=rep1&type=pdf
RP2	Early Prediction Of Merged Code Changes To Prioritize Reviewing Tasks	http://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=4991&context=sis_research
RP3	Automatically Prioritizing Pull Requests	https://pure.tudelft.nl/portal/files/67261622/vanderveenMSR2015.pdf
RP4	0-1 Programming Model-Based Method For Planning Code Review Using Bug Fix History	http://se.cite.ehime-u.ac.jp/~aman/pdf/IWESEP2013.pdf
RP5	A case study on machine learning model for code review expert system in software engineering	https://annals-csis.org/proceedings/2017/drp/pdf/536.pdf