What's bothering developers in code review?

Emma Söderberg
emma.soderberg@cs.lth.se
Dept. of Computer Science
Lund University
Sweden

Luke Church
luke@church.name
Lund University, University of Cambridge
Sweden, United Kingdom

Jürgen Börstler
jurgen.borstler@bth.se
Dept. of Software Engineering
Blekinge Institute of Technology
Sweden

Diederick C. Niehorster
diederick_c.niehorster@humlab.lu.se
Humanities Lab, Dept. of Psychology
Lund University
Sweden

Christofer Rydenfält
christofer.rydenfalt@design.lth.se
Dept. of Design Sciences
Lund University
Sweden

ABSTRACT

The practice of code review is widely adopted in industry and has been studied to an increasing degree in the research community. However, the developer experience of code review has received limited attention. Here, we report on initial results from a mixed-method exploratory study of the developer experience.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

software development, code review, user experience

ACM Reference Format:
Emma Söderberg, Luke Church, Jürgen Börstler, Diederick C. Niehorster, and Christofer Rydenfält. 2022. What's bothering developers in code review?. In *44nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510457.3513083

1 INTRODUCTION AND RELATED WORK

The utility, purpose and practices surrounding code review have been an increasing focus of research activity in recent years [6]. Studies have described the practice of code review (primarily in open-source projects [8, 9] and at big companies [2, 11]), and proposed new solutions to improve the code review process (e.g., reviewer recommendation [3], decomposition of changes [4]).

In describing the practice of code review, some studies have reported on the perception of developers, for instance, expected benefits (primarily code improvement [10], defect identification [2], and knowledge sharing [11]) and challenges (primarily understanding change rationale [2, 9–11] and code comprehension [9, 11, 12]).

However, whilst these studies report on the practitioners' perception of code review, the broader understanding of how code review fits into the socio-technical context of development is underdeveloped. A couple of studies have focused on specific aspects connected to the developer experience, for instance, perceived fairness in the code review process [8] and reasons for confusion [7]. Confusion in reviews, as well as toxic conversations [1], have also been reported as anti-patterns in code review [5].

In this work, our starting point is the developer experience of the code review process, which we see as complementary to previous work [1, 5, 7, 8]. We seek a better understanding of 'what is bothering developers when they are giving and receiving code reviews?' and input to design explorations in this space. We report on initial findings from an exploratory mixed-method study, where we focus on the use of tools and to what extent these aid developers in their code review tasks.

2 DATA COLLECTION

We used a mixed-method approach comprising 12 semi-structured interviews and a follow-up survey. We interviewed experienced developers at two multi-national businesses in Sweden. A follow-up survey was conducted to develop a broader understanding of the themes that emerged from the interviews. Rather than attempting to be comprehensive, this short paper outlines an initial exploratory analysis of the qualitative interview data to highlight emerging themes of interest after an initial reading. A more detailed analysis, including the results from the survey, is forthcoming.

3 RESULTS AND EMERGING THEMES

A recurring and novel topic that emerged from the interviews is that code review is an activity that comprises a wide range of different tasks that depend on the organisational and individual context. A single change will often be reviewed by different people in different ways. For example, the change might be reviewed by a peer for general implementation issues and other quality concerns (e.g., *"the no. 1 thing which we check for is 'are they following the coding convention?', no. 2 is if for example, if that makes sense what he/she has written?"*, developer). It might then be reviewed by a more senior developer to determine whether it aligns with the overall technical and architectural strategy of the design (e.g., *"I try not to go in to too much detail when I code review because I assume that the code there actually work and that the tests are written and so on, of course I can see it in the review as well, so I mainly focus on making*

ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05.
https://doi.org/10.1145/3510457.3513083

sure nothing has been removed and that information that would be received by the API is still there", architect).

In much of the existing literature, code review has been analyzed as a single activity [6]. However, the finding above indicates that there are differences in behavior between roles, such as between developers and architects, and levels of experience. Further from what we see in the interviews, this variation is not reflected within the tooling used for code review, which is largely the same for all use cases of the tools, as well as all roles within review. We only see an allowance for small variations to reflect social practices, for example, whether to merge a change after a single reviewer approval or after multiple.

This 'one size fits all' approach causes significant problems, and applies not only to the tooling used for the review, but to the workflow itself. For example, the reviewers who look at the code inside the wider context of architecture often need to look at a wide collection of the files. However, the set that is available within the review is defined implicitly by the collection of files that have been modified and can not be changed by the developer submitting the review.

We see a **mismatch between the 'Unit of Analysis' and the 'Unit of Attention'**, that is, a mismatch between the scope of data that is presented to the reviewer (Unit of Analysis) and the scope of the system that the reviewer needs to attend to (Unit of Attention). Whereas the Unit of Attention varies depending on the type of change, the nature of the review, and the role of the reviewer, the Unit of Analysis is always the same, since it is implicitly defined by the change set. The reviewers have to mitigate this problem by manually adjusting the Unit of Analysis, e.g., by switching from the review tools to their IDE to load further code for review (*"if it is a very complex change or a change that I'm not very familiar with, then I usually check out the commit and then I have a look at it in my IDE where I have a better syntax highlighting and can follow code more easily"*). This tool-task misalignment can cause a significant interruption of the reviewer's workflow and a distraction from the task at hand, which might create the risk of them taking a shortcut and not performing the review at the thoroughness they otherwise would.

4 DISCUSSION AND FUTURE WORK

The 'one size fits all' approach provided by current code review tooling does not support developers' and reviewers' behaviour in the wild, and causes time-consuming workarounds to realign the unit of analysis and attention.

There are two potential strategies to address this. One would be to explicitly support roles within the review tools, so that, for example, an architectural reviewer might see a different presentation of the review and source code compared to an early reviewer concentrating on the detail of the code. Whilst this would address the problem, it would require extensive user-centered design work in order to characterise the needed roles and design suitable presentations for each.

An alternative strategy would be to increase the flexibility of the tools and allow developers and reviewers to curate the set of content being reviewed, and allow teams to develop their own practices. Whilst this would not require the upfront work to characterise the different roles and practices, it would require more work from the developers and reviewers to curate the review sets and develop their own practices.

Both approaches seem viable routes for creating a more adaptive approach to code review that might address the problems caused by the existing uniform approach.

ACKNOWLEDGMENTS

This work has been partially supported by ELLIIT - the Swedish Strategic Research Area in IT and Mobile Communications, the Swedish Foundation for Strategic Research (grant no. FFL18-0231), and the Swedish Research Council (grant no. 2019-05658).

REFERENCES

[1] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and M.A. Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Information and Software Technology* 114 (2019), 37–54. https://doi.org/10.1016/j.infsof.2019.06.005

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. 712–721. https://doi.org/10.1109/ICSE.2013.6606617

[3] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. 931–940. https://doi.org/10.1109/ICSE.2013.6606642

[4] Mike Barnett, Christian Bird, João Brunet, and Shuvendu K. Lahiri. 2015. Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, Vol. 1. 134–144. https://doi.org/10.1109/ICSE.2015.35

[5] M. Chouchen, A. Ouni, R. G. Kula, D. Wang, P. Thongtanunam, M. W. Mkaouer, and K. Matsumoto. 2021. Anti-patterns in modern code review: Symptoms and prevalence. In *Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering*. 531–535. https://doi.org/10.1109/SANER50967.2021.00060

[6] N. Davila and I. Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software* 177 (2021), 110951. https://doi.org/10.1016/j.jss.2021.110951

[7] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik. 2021. An exploratory study on confusion in code reviews. *Empirical Software Engineering* 26, 1 (2021), 1–48. https://doi.org/10.1007/s10664-020-09909-5

[8] Daniel M. German, Gregorio Robles, Germán Poo-Caamaño, Xin Yang, Hajimu Iida, and Katsuro Inoue. 2018. "Was My Contribution Fairly Reviewed?": A Framework to Study the Perception of Fairness in Modern Code Reviews. In *Proceedings of the 40th International Conference on Software Engineering (ICSE) (ICSE '18)*. ACM, New York, NY, USA, 523–534. https://doi.org/10.1145/3180155.3180217

[9] O. Kononenko, O. Baysal, and M. W. Godfrey. 2016. Code Review Quality: How Developers See It. In *Proceedings of the 38th International Conference on Software Engineering (ICSE) (ICSE '16)*. ACM, New York, NY, USA, 1028–1038. https://doi.org/10.1145/2884781.2884840

[10] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka. 2018. Code Reviewing in the Trenches: Challenges and Best Practices. *IEEE Software* 35, 4 (2018), 34–42. https://doi.org/10.1109/MS.2017.265100500

[11] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. 2018. Modern Code Review: a Case Study at Google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 181–190. https://doi.org/10.1145/3183519.3183525

[12] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli. 2018. When Testing Meets Code Review: Why and How Developers Review Tests. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. 677–687. https://doi.org/10.1145/3180155.3180192