Understanding the Experience of Code Review: Misalignments, Attention, and Units of Analysis

Emma Söderberg
emma.soderberg@cs.lth.se
Lund University
Lund, Sweden

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

Jürgen Börstler
jurgen.borstler@bth.se
Blekinge Institute of Technology
Karlskrona, Sweden

Diederick C. Niehorster
diederick_c.niehorster@humlab.lu.se
Lund University
Lund, Sweden

Christofer Rydenfält
christofer.rydenfalt@design.lth.se
Lund University
Lund, Sweden

ABSTRACT

Code review is a common practice in software development and numerous studies have described different aspects of the process; its characteristics, the expectations on that process, issues around reviewer allocation, and more. However, one aspect that has not been studied to a large extent is the experience of the developers in the code review process. This is unfortunate given the significant amount of time that developers spend on this activity, where problems that degrade developers' experience on a daily basis can create work environment issues.

In this paper, we present an extended analysis of an exploratory mixed-method study where we focus on developers' experience of code review. We use semi-structured interviews to gather data from two multi-national companies and conduct a follow-up survey. Our results suggest that developers are frequently bothered by misalignments in the code review tooling and process which is hindering them in carrying out their code review tasks effectively. We present an initial characterization of misalignments that may hamper the developer experience. Based on our findings, we propose directions for further exploration to improve 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. Understanding the Experience of Code Review: Misalignments, Attention, and Units of Analysis. In *The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022), June 13–15, 2022, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3530019.3530037

EASE 2022, June 13–15, 2022, Gothenburg, Sweden

ACM ISBN 978-1-4503-9613-4/22/06.
https://doi.org/10.1145/3530019.3530037

1 INTRODUCTION

Code review, where developers review each other's code changes via tools before they add it to the code base, is a common practice in the software industry [1, 8]. Empirical research consistently shows that modern (tool-supported) code reviews have a positive impact on software quality [15, 16, 18, 19, 23]. However, there is also evidence showing that code review requires a lot of effort [3, 15, 20], might not always be effective [8], can induce confusion [10], and can harbor toxic conversations [6] and bullying [4].

Developers are central to the process of code review, but reporting on their experience in code review is nascent. Previous studies have summarized difficulties in code review [1, 14, 15, 18, 21], understanding code [1, 18] and timely feedback [15] as primary challenges. The report by Sadowski et al. [18] includes causes of breakdown in the code review process leading to frustration, e.g., the tone and use of power in code review communication, or the topic of the review and whether the code review tool is the best place to conduct a design review. Related to these breakdowns, recent work by Chouchen et al. [6] analyzed past studies on code review and presents five anti-patterns in code review; confused reviewers, divergent reviews, low review participation, shallow reviews, and toxic reviews. Delving deeper into confusion, recent work by Ebert et al. [10] studies confusion in code review; reasons for it, the impact of it, and coping strategies.

While this existing work provides interesting insights into areas where the code review process likely affects the developer experience, we see an opportunity for further qualitative investigations to complement and expand on this work.

In this paper, we focus on developers' practise and their experience of using tools in the code review process and to what extent these aid them in the code review tasks. With this endeavour we seek design input for possible tool-oriented interventions in the code review process to improve the digital work environment of software developers. Like in other domains, it seems reasonable to assume that many small interaction issues may compound to have larger negative effects on the digital work environment [13].

The interviews were analyzed thematically [6] as an extension to an initial reading of the interview notes and transcripts [1].

We report on an exploratory mixed-method study, where we expand on an earlier initial reading of a series of interviews of developers, technical leads and managers carried out at two multinational companies [22] that develop products with significant software components with a thematic analysis [5]. In this expanded analysis, we identified themes centered around the experience of the developers and misalignment between the code review task and the tools or processes. We followed up with a survey, focused on the misalignment between the code review task and the tools, and found this to be a wide-spread phenomenon, reported as occurring in every fourth code review on average. Based on this observation and others, we propose directions for future exploration of designs of processes and tools. The contributions of this paper, beyond [22], are as follows:

- a preliminary characterisation of misalignments affecting the developer experience in the code review process.
- suggestions for future explorations focused on improved developer experience in the code review process.

The remainder of this paper is structured as follows; we start with an overview of related work in Section 2 before we cover methodology in Section 3, then we go through the results in Section 4 and discuss these in Section 5, cover threats to validity in Section 6 finally and conclude in Section 7.

2 RELATED WORK

A mapping study by Badampudi et al. [2] covering research on modern code review 2005–2018 shows that process and tool improvements, impact and outcome, and reviewer selection are the top-3 research areas in the field during this period. A recent mapping study by Davila and Nunes [9] further proposes a taxonomy of the research on modern code review. In their taxonomy, the work presented here falls into two categories[1]: (1) foundational studies, i.e. works that try to understand practical aspects of modern code reviews, more specifically, subcategory practitioners' perceptions of the state-of-practice and (2) evaluations, more specifically, subcategory opinion studies. Davila and Nunes summarize challenges and difficulties from papers covering this topic [1, 4, 14, 15, 18, 21], listing code comprehension as the main challenge, followed by difficulties with time pressure and tool support.

Bacchelli and Bird [1] present a study at Microsoft focused on learning more about motivations, expectations, and challenges in code review, gathering data via observations, interviews and a survey. The main challenge found for reviewers was understanding the change under review from the provided information, seeking different ways to gather missing information: reading the description, running the code, face-to-face communication. The authors highlight a lack of support for program comprehension in code review tools.

Kononenko et al. [14] present a study based on a survey on developers' perception of code review quality. The main challenge found for developers when performing review tasks was to "gain familiarity with the code" under review and in relation to this also code complexity and size. Personal challenges concerning time management, improving technical skills, and context switching between tasks are also mentioned. The respondents in the study were asking for better support for getting the code under review into the editing environment from the Bugzilla review tool and for better support for code browsing in the code review tool.

Baum et al. [4] applied grounded theory to understand why code review is used and why it is used in a certain way. They report on desired and undesired effects in code review, listing authors being offended and occasionally bullied as an undesired effect.

MacLeod et al. [15] present a study at Microsoft, with observations and semi-structured contextual interviews followed by a survey. They find several challenges in the code review process, presented from the author perspective (e.g., receiving timely feedback, receiving insightful feedback) and the reviewer perspective (e.g., difficulty managing large reviews, finding time to do reviews). They then match challenges to best practise as recommendations to mitigate a challenge, for instance that authors select reviewers with the right expertise, or let reviewers self-select their review tasks to counteract the reviewer challenge of understanding a change. Their recommendation regarding tools is to ensure that appropriate tools are provided on the organisational level.

Sadowski et al. [18] studied code review at Google, gathering data via interviews, a survey, and logs analysis. Challenges for developers are reported in terms of breakdowns and listed (after acknowledging that understanding is a challenge as reported in [1]) as four breakdowns; in the process (organizational and geographical distance, social interaction regarding tone and power, review subject, expectations on change context), and one related to the tool (customization of required approvals).

Spadini et al. [21] investigated how developers review tests based on mining of code reviews followed by interviews. They found that review of test code is different from review of production code, and that lack of code navigation is an issue for developers.

In relation to the works listed above [1, 4, 14, 15, 18, 21], our work has a stronger focus on the developer experience and to what extent the code review tools are supporting code review tasks. Our focus on experience is closer to the recent work by Chouchen et al. [6] and Ebert et al. [10].

Chouchen et al. [6] present anti-patterns in code review gathered from reviewing the literature; confused reviews (e.g., reviewers ask questions about rational for a change), divergent reviews (e.g., no consensus in reviewer decision), low reviewer participation (e.g., few comments or lack of prompt feedback), shallow review (e.g., shallow comments on complex changes), and toxic review (e.g., comments with a negative sentiment). They manually inspect a sample of 100 code reviews and find a presence of all anti-patterns in the sample. While the existence of anti-patterns is interesting, more research is needed in order to understand how and why they occur, and an important part of that puzzle is likely related to the experience and perspectives of the reviewers themselves. As such, our study is complementary to the work by Chouchen et al., and specifically the methods we use are different; we start out with qualitative data gathering via interviews and follow up with a survey to investigate broader occurrence of themes constructed from the interview data.

[1]Davila and Nunes [9] define 3 main categories comprising 7, 3, and 4 subcategories respectively.

Table 1: Interview Participants (S1 for series 1, and S2 for series 2 participant), their role, experience, and code review tools used.

Participant	Company	Role / Experience
S1-1	A	Developer / Senior (10+y)
S1-2	A	Developer / Junior (<1y)
S1-3	A	Developer / Tech lead (10+y)
S1-4	A	Developer (10+y)
S2-1	B	Developer / Design lead (5+y)
S2-2	B	Developer / Software designer (20y)
S2-3	B	Manager / Scrum master (20+y)
S2-4	B	Developer (10+y)
S2-5	B	Developer / Architect (5+y)
S2-6	B	Developer / Team lead (5+y)
S2-7	A	Developer (5+y)
S2-8	A	Developer (10+y)

Ebert et al. [10] studied confusion in code review. They investigated the reason for confusion, the impact of that confusion, and the coping strategies developers use. They survey developers for reasons for confusion and analyze code review comments for sources of confusion. They follow-up with a second survey to investigate how common the found reasons for confusion are. Finally, they surveyed the literature to investigate to what extent the top five reasons (long and complex changes, organisation of work, dependencies between changes, lack of documentation, and missing change rationale) were mentioned. While Ebert et al. reveal multiple reasons for confusion, it is a cognitive aspect of the interaction related to the reviewer. As such confusion does not capture the full phenomenon of the interaction between reviewer and code including its context. Thus, we also need to explore the reviewers' experiences of working with code review in such way that both the context and intention associated with their code review practice is considered.

3 METHODOLOGY

We carried out an exploratory mixed-method study with subjects from two multi-national companies (A, B) with established code review practices. We started with a series of semi-structured interviews to learn more about the developers' experience of code review, and conducted a follow-up survey to broaden our understanding of the phenomena seen in the interviews.

Interviews. We gathered data via two series of interviews with software developers. The first series consisted of 4 interviews at company A and the second series of 8 interviews at company A and B. In total 12 interviews were conducted. The interviewees had between 0–20 years of experience of software development, with all but one having more than 5 years of experience (see Table 1). We recruited interviewees by reaching out to contact persons from the network of the authors and the contact persons then connected us with employees at the companies. All interviews were carried out over video calls. For the first series of interviews we took notes, while for the second series of interviews, we also recorded the sessions after informed consent from the participants. The second author conducted all the interviews with the first author taking notes during all interviews. In the first series of interviews, we had an additional note-taker. Table 1 lists the self-described roles of the participants and their experience with software development (referred to as S1-1 to S1-4 for the first series and S2-1 to S2-8 for the second series of interviews). In preparation for analysis, the recorded interviews were transcribed, this work was carried out by the first, second, fourth and fifth author.

The interviews were analyzed thematically [5] as an extension to an initial reading of the interview notes and transcripts [22]. As an initial analysis, three authors (first, second, fifth) read through all transcripts separately, looking for interesting things that stood out in the transcripts. All authors met to discuss potential themes from this bottom-up read-through and we discussed and unified around four initial overall themes in the data. The first and second author then read through the transcripts top-down connecting quotes to these themes, with an overlap of one transcript. The coding of this overlapping transcript was used by the first and second author to unify and iterate on the interpretation of the themes. The coding of the themes was sent to the fifth author for review, in preparation of a final reflective discussion between the first, second and fifth author where the four initial themes were unified into two overall themes, which both consisted of multiple sub-codes that represented different types of observed misalignments.

Survey. The analysis of the interviews showed that interviewees to a large extent mentioned misalignments between the review task and the tools used. We decided to follow-up this finding by a survey to explore to what extent this misalignment occurs beyond our interview sample. The survey was developed by the third author using the tool Unipark[2] and run in parallel with the analysis of the interviews. Data was collected during 2 weeks in the Fall of 2021. Invitations were sent to a wider sample at company A and B via dedicated contact persons reaching ~400 employees at company A and 680 employees at company B.

To avoid bias, survey respondents were first asked general questions about perceived benefits and obstacles with code reviews. After that, we described the concept of tool-task misalignments in general terms and inquired about their experience with such misalignments, if they had any.

The free-text answers from survey questions that were directly or indirectly concerned with misalignments (Q5, Q6 and Q8; see Figure 2) were coded by the first and fifth author separately using the code book developed from the interviews. The first and fifth authors then compared their coding and discussed it until they reached an agreement on the coding.

4 RESULTS

In this section, we presents the results of the interviews (Section 4.1) and the follow-up survey (Section 4.2).

4.1 Interviews

We constructed two main themes centered around misalignment in the code review process and its tools: 1) between the code review tool and the needs of the code review task for a reviewer (*Tool-Task Misalignment*), and 2) between the code review process and the

[2]https://www.unipark.com

code review task of a reviewer (*Process-Task Misalignment*). Both themes are presented in more detail below.

4.1.1 Theme: Tool-Task Misalignment. In this theme, we are considering misalignments we saw between the task a reviewer was trying to carry out and the extent to which this task was supported by the code review tool. Participants mentioned cases where the conversation facilitated by the code review tool (asynchronous communication via comments on a change) did not support the conversation needed by the review task, e.g., due to complexity (*"when things are particularly complex or delicate, then it usually makes sense to have a white board session about it"*, S2-2) or sensitivity (*"if there's any critical feedback in the code, I think many people become defensive. It is better to resolve that in a meeting I think."*, S2-3).

Within these conversations, we also saw a lack of support for means of communication for providing information needed by the conversation (when carried out in the code review tool). For instance, lacking support for expressing links to resources beyond the diff of the change (*"a change somewhere that's affecting some other part of the code which just doesn't happen to be part of the patch set, then I'm not able to give comments over there, so that's, then you start to make references on your comments"*, S2-2). Similarly, a front-end developer in need of communicating the effect of a change in the form of videos or images could not easily include this in the metadata of a change (*"we try to include like videos or images as well. ... that sort of discussions can easily occur in those tools, I think. But but like on Gerrit, I don't think you can include like images that at least show up in that user interface so"*, S2-8).

Finding 1: We saw **tool-conversation misalignment**, where the kind of conversations needed for a review task and the kind of conversations facilitated in the code review tool were misaligned.

Several participants further mentioned that for some changes they could not find the information they needed in the code review tool. This lack of information would typically cause developers to leave the review environment, by pulling down the change to their editing environment, to find the information that they were missing, e.g., assessing flows in the code (*"It's just when I'm unsure if it works or not, I cannot see the flow of how it would work so I try to try it out, so what if this scenario happens what would happen then, I can't see that in the flow"*, S2-5), or effect beyond the change (*"I can see that this could have a negative impact on other things, or I can't accurately get a sense of what it does because it touches so many different places. If any of this is true I usually pull it down and take a better look at it"*, S2-6). Change characteristics such as complexity and unfamiliarity were mentioned as triggers for a move to the editing environment (*"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"*, S2-1).

In reference to the editing environment, the customized code formatting in the editing environment (*"it highlights in a different colour than my eyes are used to"*, S2-4) together with code browsing support (*"To fire it up in the code editor as well so you can click around and use the code navigation"*, S2-8), were mentioned as helpful features for the review task. The possibility of experimenting with the change, to gather data for a modification suggestion in a review comment, was also mentioned as something that was possible in the editing environment (*"You have to try a couple of things if you want to suggest any big changes in the change request."*, S2-4).

We further saw that reviewers were looking for different information in the review, that is, that they had different 'units of attention'. For instance, a developer mentioned checking coding standards (*"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?"*, S2-4), while an architect mentioned focusing specifically on an API at the later stages of a review of a change (*"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 sure nothing has been removed and that information that would be received by the API is still there"*, S2-5).

Finding 2: We saw **tool-information misalignment**, where the information reviewers needed for a review task and the information available in the code review tool were misaligned. Participants would typically resolve this misalignment by leaving the code review tool in favor of the editing environment.

Several participants mentioned the size of changes and how the suitable size was to some extent mandated by the process, advocating for smaller changes (*"We also try to work in an agile way, so that we keep changes as small as possible. That does not mean that they are always small"*, S2-7). At the same time, getting changes in a shape that is compliant with the mandated size was described as a something that can be challenging (*"it's not always easy to slice up your code into 300 lines of changes that make sense"*, S2-8).

The size of a change frequently came up in the interviews as central to how the developers worked with the code review tool and process. Size was mentioned as a key predictor for whether a developer would pull down the change for further reviewing in their local environment (*"if the change request is huge, often we check out that change request and run it on our local.. and just try to understand what he or she has done"*, S2-4). The experience of reviewing a large change was described in negative terms (*"the bigger the patch sets the more difficult to review it, the more time it takes. It's almost exponential"*, S2-6).

Size was also connected to tool performance, as in the tool becoming impractical and slow for changes beyond a certain size (*"to look at change lines of code is not always that easy, and if you have large change sets, it could be horrendously slow to run their diff view for example in the browser"*, S2-7). In one case, the performance of the tool was even mentioned as being used as a kind of test to check if the change should be broken up into smaller pieces (*"When it comes to actually reviewing the code changes I mainly do that in the browser, because we feel that if that is not doable anymore, then maybe it has become too big and you need to break it down"*, S2-7).

Finding 3: We saw **tool-size misalignment**, where the size of a change and the size suitable for review were misaligned.

4.1.2 Theme: Process-Task Misalignment. In this theme, we consider the coordination of the code review process, between the individuals in the review and between the organization and the

process. For instance, developers described a team practice where the review load is evenly shared within the team (*"It's the team practice, everyone has to review the code of others"*, S2-4), but the actual review work is not shared evenly in practice (*"even though it's like everyone is responsible, not everyone does"*, S2-1).

We also saw this implicit assignment of review tasks beyond the internal team practice, for instance, one architect described a process where review tasks would manually be inspected and self-selected for review (*"I'm not really asked to review it, it's kind of like 'for your information' … I tend to just, yeah, when I look I just tend to inspect it whether I should review this or no"*, S2-5).

We also saw a misalignment in who is responsible to follow up on comments given in the review, where a system designer described it as being up to the author in some cases (*"sometimes you are just sharing your thoughts and your opinions on things and leave it up to the patch set author to decide whether or not he will bring it up or not or make that change or not"*, S2-2) and an architect described a team practice where the reviewer is responsible (*"if you have a lot of people reviewing the same patch and the author just keeps pushing new stuff, what tends to happen is that some of your reviews get forgotten, your comments get forgotten, so you have to follow it up, you have to follow up your own comments, it's kind of one of the rules we have as well, that is you post a comment on a patch it's your responsibility to follow up on that comment as well, to make sure the author has seen it and has taken some kind of action"*, S2-5).

Finding 4: We saw **process-responsibility misalignment**, where the person expected to do a task in the review and the person actually performing the task were misaligned.

Participants further conveyed frustration from both the author side (*"if you want something reviewed it's kind of, it can take a while before it gets reviewed I think that's the most frustrated part"*, S2-5) and the reviewer side (*"when it comes to this massive patch set I won't be able to just look at the code and see how it all works. That definitely causes frustrating as it takes much more time from me to do it."*, S2-6) about when review tasks were not completed within the expected time.

The risk of blocking colleagues in the review process was described by one developer as a motivation for doing reviews (*"I feel like doing your reviews is a way to unblock my colleagues, right, making sure that nobody is just sitting there waiting, and I hope people will do the same for me in the end because I just hate it when things are unreviewed for days"*, S2-2).
A team lead involved in most reviews on the team described a common use of side-channels to unblock the code review process, and remorse for having to be reached via these side channels on a regular basis (*"Sometimes when I'm very busy I don't see the mail, someone will ping me and say 'can you look at this patch'?"*, S2-6).

The manager we interviewed described the code review process with more focus on the people in the process rather than the tools (*"It's more of a people's problem, getting people involved"*, S2-3).

Finding 5: We saw **process-completion misalignment**, where the expected time for a review task and the actual time for completing a task were misaligned.

Some participants mentioned that the individual style for how an author writes code may collide with the mandated style of automated review comments and how these are configured. One developer described an initial reluctance to the introduction of automated style comments that changed to a sense of relief when discussions of style could reduced in the review (*"I was sort of against it at first, I think, but it's been hugely like liberating actually, cause, you don't have to think anything about you know how the code is formatted and or and you don't have to review that stuff either. Yeah, it just happens and it doesn't look like super nice all the time, but yeah, you don't have to care about that right"*, S2-8).

Another developer described a view where the authors should have freedom to write code in their own style beyond the line drawn by the automated review tools (*"as long as you follow what the linter and the formatter does, I am like, then you should let the person have, not try to change their code to what you would have written. Cause then I think you're on the wrong track with your review, then you are like rewriting someone else's stuff like you would have done it"*, S2-7), implicitly accepting the automated style recommendations, while another developer expressed doubt about the style being manifested by the automated comments (*"What configurations are good, what configurations are bad? Those preloaded configurations for IDE are based on someone's experience. We can't claim that his experience, or her experience, is the only good thing or the only right thing"*, S2-4).

Finding 6: To some extent we saw **developer-automation misalignment**, where the expected automation in the process and the actual automation taking place were misaligned.

One developer expressed concerns about how new developers are being exposed via the code review tool (*"Gerrit can be very public and if you are a person with insecurities then it can be bad"*, S2-1), while the manager we interviewed expressed no concern with the tool (*"I think the tool works well, I don't see any problems with Gerrit."*, S2-3). When asked about developers pulling down changes in order to review, the manager had not seen this behavior (*"I'm not sure why they would need to do that [leave the code review tool]. Maybe they feel they can't review the code because they're not experienced in this area, and so they need to play around a little to understand how the code is to give proper reviews. I have not seen that"*, S2-3).

Finding 7: We saw **developer-experience misalignment**, where the expected experience by participants of the code review process and the actual experience were misaligned.

4.2 Survey

Of the ~1080 employees who received an invitation, 119 opened the survey. Of those 119, 78 started answering the survey and 44 completed it. The results presented below are based on all available data, i.e. n=[44..78].

4.2.1 Participant background. The survey results show that most of the respondents have different developer roles, with 68.2% of respondents identifying themselves as developers. The remaining 32.8% identified themselves as system architects (13.6%), team leaders (11.4%), testers (2.3%), or "other" (4.6%). No-one identified him- or herself as a manager or a researcher. They also have substantial

practical experience as software developers and with code reviews (Q1 and Q2 in Table 2) and spend 6.2 hours per week on average on code reviews with a median of 1 hour per day for an average work day (Q3 in Table 2). The answers to the questions about code review processes and tools suggest the existence of established code review processes (Q15, see Table 3).

Table 2: Experience of survey participants (n=76).

Aspect	min	max	avg	med	stdev
Years of practical experience as prof. software developer (Q1)	1	30	11.3	10	7.6
Years of practical experience with code reviews (Q2)	1.2	27	8.8	7	6.1
Hours per week on average spent with code reviews (Q3)	0.5	20	6.2	5	4.4

Table 3: Level of agreement with statements about code review processes and tools (Q15, n=44). Agreement is on a Likert scale from 1 (completely disagree) to 5 (fully agree).

Statement	Disagree (1-2)	Agree (4-5)
My team follows a commonly agreed process for code reviews	18.2 %	77.3 %
My team's code review process is supported by tools	9.1 %	77.3 %
My team's code reviews are fair and objective	6.8 %	88.6 %
Without tools thorough code review would not be possible	27.3 %	63.6 %

4.2.2 Perceived benefits of code reviews. We asked respondents to rank, according to their personal experience, a number of statements regarding presumed benefits of code review (Q4). The results, listed in Table 4, show that "improved software quality" was ranked highest followed by "improved knowledge sharing"; "reduced time to market" and "improved customer satisfaction" were ranked at the bottom. The top rankings of knowledge sharing and software quality are in agreement with what previous studies have found regarding the expected benefits from code reviews [1, 18].

Table 4: Ranking of presumed benefits of code reviews (Q4, n=78). The n/a column lists the number of responses not mentioning this benefit.

Presumed benefit	avg	med	stdev	n/a
Improved overall software quality	1.83	1	1.21	3
Improved knowledge sharing	2.87	2.5	1.58	8
Improved collaboration	3.55	3	1.63	16
Improved ability to integrate new or less experienced team members	3.66	4	1.33	7
Improved adherence to common standards and guidelines	3.72	4	1.81	11
Reduced overall project costs	5.4	5	1.68	36
Improved customer satisfaction	5.71	7	2.34	44
Reduced time to market	6.87	7	1.31	47

4.2.3 Perceived review obstacles or challenges. After asking about benefits, we asked respondents to describe review obstacles or challenges. To avoid bias, the first of these questions (Q5) was asked before the concept of tool-task misalignment was introduced. Questions Q6–Q11 asked about respondents' experience with tool-task misalignments, if they had any. The survey questions were presented in a way that respondents could neither see any mention of the concept up until and including Q5, nor could they go back to Q5 or earlier questions after the introduction of the concept.

Our results show that respondents experienced tool-task misalignments in about every fourth code review on average (see Figure 1).

Finding 8: Survey respondents experienced tool-task misalignment in every fourth code review on average.

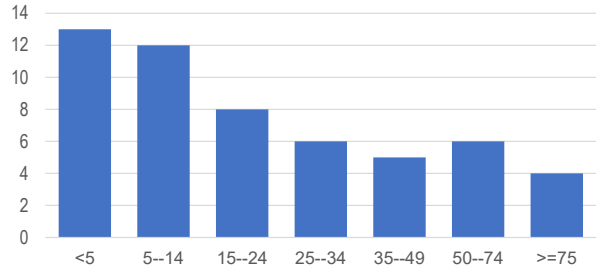

Figure 1: Percentage* of typical code reviews where respondents experienced tool-task misalignments; avg 24.5%, median 17.5% (Q7, n=54).

*The grouping into ranges is by the authors. Responses were in terms of specific approximate numbers that were elicited using a slider from 0–100.

Respondents' free-text descriptions of review obstacles or challenges (Q5) and experience with tool-task misalignments (Q6 and Q8) were coded with the misalignment codes from Section 4.1 to facilitate comparison of interview and survey results. Figure 2 gives an overview of all codes for survey questions Q5, Q6 and Q8 for each misalignment code (leftmost boxes), and then splits them based on respondents code review experience, role, and hours spent in code review each week (from left to right).

In the responses to Q5, we note that the respondents had several things on their mind, with process-completion (Proc.-Comp.) being the most dominant code, regardless of how we divide the coded answers on code review experience, role or hours in code review per week. In the left-most scatter plot, we see tool-information (Tool-Info) as the second most used code followed by tool-size (Tool-Size), and they are present to some extent for all data division variants. The tool-size code stands out as an obstacle or challenge that almost all team leads are concerned about.

Finding 9: All misalignments seen in the interviews appear in the survey responses. The top three misalignments reported by survey respondents were process-completion, followed by tool-information and then tool-size.

When further analysing the responses to Q5, we saw additional focus on lack of knowledge (related to tool-information) about a code base (e.g., *"It's hard to carry out code reviews if you have no previous understanding of the underlying code that is being modified*

Figure 2: Observed misalignment codes in answers to free-text survey questions Q5, Q6, and Q8. The questions are included verbatim below the question numbers. The misalignment codes to the right of the question texts correspond to the 7 highlighted misalignments discussed in Section 4.1 (e.g., Tool-Conv. refers to tool-conversation misalignment). The numbers in parentheses under each scatter plot column show the total number of answers for the respective group (e.g., Q5 had 75 answers in total shown under "Codes", and 29 answers from respondents classifying themselves as developers under "Codes/Roles"). Bubble sizes are relative to the totals of a group (e.g., 12 developers of 29 mentioned process-completion for Q5).

or added to" and *"The largest obstacle/challenge is when you are unfamiliar with the code base. Then it is difficult properly to put the code you are reviewing in to context with the rest of the code. Something that looks fine might break something else which is perhaps something a more experienced person would have caught."*).

We further saw answers related to developer-experience (Dev.-Exp.) where respondents would express negative emotions about the review process in a way that we did not see in the interview study, e.g., lack of interest (*"Lack of technical interest in the code under review makes it hard to provide good feedback."*), code review being boring (*"It is extremely boring and monotone"*), discomfort with giving critique (*"They are monotonous and repetitious. Not as creative as other part of software development. You get the feeling that you are always complaining when you are leaving comments"*).

Finding 10: Survey respondents mentioned lack of knowledge of the code base and lack of engagement, due to code review being perceived as boring and monotone, as obstacles to carry out code review tasks.

After introducing the concept of tool-task misalignment (before Q6), we can see that tool-information (Tool-Info) dominates respondents' answers in any way we split the data. Almost all team leads mentioned the tool-information misalignment in their answers.

When asked for "how much or little do tool-task misalignments disturb you on average when they occur while carrying out a code review" on a scale from 1–7 (Q10), 60.4% answered "not disturbing" (1–3), 22.6% answered "disturbing" (5–7) and 5.7% that they do not experience tool-task misalignments (n=53).

We continued with a question where we asked respondents to *"describe the main reasons for the tool-task misalignments that you experience"* (Q8). The bottom row in Figure 2 shows the results of coding the free-text responses for survey question 8 (Q8). Again, tool-information dominates the coding. Interestingly, the team leads that displayed a substantial agreement earlier for Q6 do not stand out any more with regard to any code.

Finding 11: Tool-information misalignment stands out as the main tool-task misalignment in the survey responses, where 22.6% report finding tool-task misalignment disturbing.

We asked respondents about how they typically handle a tool-task misalignment (*"What do you typically do to resolve a tool-task misalignment"*) and found an overarching trend of leaving the code review tool in this situation, either seeking a different kind of conversation (e.g., *"Talk on a different tool about the review"* or *"Pasting screen shots and comments in a mail instead of using a review tool"*), or seeking information available in the IDE (e.g., *"For complex reviews, I sometimes need to download the change and open it in my IDE to get the full view"* or *"Mostly, I load the code into IDE and explore dependencies or check completeness there"*).

Finding 12: The survey respondents would primarily leave the code review environment to resolve a tool-task misalignment, either to seek a richer communication channel or more information available in the IDE.

5 DISCUSSION

We set out to study the developers' experience of the code review process, starting with a series of interviews. In our analysis of the interviews we find several misalignments; the code review tool may not support the kind of conversation needed by the code review task (tool-conversation, **Finding 1**), it may not provide the information needed for a review (tool-information, **Finding 2**), and the size of a code change may not align with the size suitable for review (tool-size, **Finding 3**). We recognize some aspects of these misalignments from the literature, for instance, Ebert at al. [10] report 'long or complex code changes' and 'organization of work' as top reasons for confusion which connect to tool-size and tool-information. They further report that 'off-line discussion' is used as a coping strategy to resolve communication issues which relates to tool-conversation. We further see that tool-information misalignment is connected to the previously reported challenge of code understanding [1, 18] and to previously reported information needs of reviewers in code review [17]. However, we have not seen any extensive report on how developers are managing tool-information misalignment. The closest we have seen is in a brief mention of the behavior by Spadini et al. in their study on review of tests [21] (*"That's why I pull the PR every-time and I lose a lot of time doing it"*, p. 685).

We further see in the interviews that misalignment in expectations in the code review process may cause frustration, in that the responsibility of review tasks may not align with who actually performs tasks (process-responsibility, **Finding 4**), and the completion of review tasks may not happen when expected (process-completion, **Finding 5**). This connects to the work by Ebert et al. [10], who report 'organization of work' as a top reason for confusion, connecting to process-responsibility (but their focus is on status of changes, e.g., ready to review?). They further also report delays as a possible impact of 'organization of work' which connects to process-completion. MacLeod et al. [15] also mention lack of timely feedback in code review as a challenge.

Further in the interviews, we saw a misalignment between a developer's expectation on automation in the code review process (developer-automation, **Finding 6**), as well as, the expected experience of the code review process (developer-experience, **Finding 7**). In the case with automation, the code review tool comes across as a shared space where individual preferences and the norms of the group meet, and where challenges to the norm may result in disagreements. In the case with expected experience, participants highlight the experience of getting into the code review process which can be harsh for developers new to the process, especially as the process involves being criticized in a public space. In both these cases, we see little in the literature on these topics and see them as possible directions for future work.

In the follow-up survey focused on the tool-task misalignment theme, we saw all misalignment codes from the interviews appear in the survey data (**Finding 9**), with process-completion being the most dominant code. We note that this occurrence of misalignment appears in responses to question Q5 on obstacles, which was asked before the concept of tool-task misalignment was introduced to the survey respondents. We additionally see two aspects of the code review experience appearing in the survey responses and not in the interviews; lack of knowledge of the code base and lack

of engagement due to code review being perceived as boring and monotone work (**Finding 10**). These obstacles relate to the work by Ebert at al. [10], reporting 'lack of familiarity with existing code' as a reason for confusion, and to the work by Chouchen et al. [6], where 'low reviewer participation' and 'shallow review' are listed as code review anti-pattern.

When we ask specifically about tool-task misalignment in the survey, respondents report that they experience this in in every fourth review on average (**Finding 8**), with 22.6% of respondents reporting this as a disturbance (**Finding 11**). The responses further primarily mention tool-information misalignment as the cause (**Finding 11**). When asked about how they resolve tool-task misalignments the answers were focused on resolution of tool-information misalignment and moving to the editing environment to find missing information (**Finding 12**) which aligns well with the behavior mentioned by interview participants (**Finding 2**).

In the following paragraphs, we present reflections along with suggestions for further exploration into the design of processes and tools, expanding on an initial reading of the interviews [22].

Reflection 1. **Unit of Analysis vs Unit of Attention** When reflecting on tool-information (**Finding 2**) and tool-size misalignment (**Finding 3**), we note that when developers (both authors and reviewers) interact via a code review tool, the code is packaged into a single unit of analysis. In both Gerrit and GitHub, this unit of analysis is implicitly derived from the lines of code that have changed. This is different from the unit of attention, that is, where developers need to look in order to review the code. Whilst this attention is correlated with the changes, and by necessity starts there, it often doesn't align perfectly. Sometimes there are cases where many changes aren't meaningful for developers to review (e.g. lists of hashes), but more commonly, the reviewer's attention needs to go beyond the context of the unit of analysis presented to them by the review tool causing them to have to exit the tool to see the wider codebase.

Suggestion 1: explore process and tool designs for better alignment of the 'unit of attention' and the 'unit of analysis'. It may be worth considering whether missing information should be brought into the code review tool, or if the code review task should be brought to the information (e.g., into the IDE).

Reflection 2: **Task Coordination** When considering the unit of analysis and the unit of attention together with tool-conversation (**Finding 1**), process-responsibility (**Finding 4**) and process-completion (**Finding 5**), we see a further possible misalignment when the reviewers want to play a particular role in the review. For instance, not reviewing all of the code but only looking at the big picture, assuming for example that the code-level changes have been taken care of by others. Sometimes the developers of the code try and assist the reviewer by commenting where they should look – trying to help them focus their attention on a smaller unit of analysis than the tool provides, but this is mostly done in an ad hoc way using secondary notations [11] and without explicit support in the tool.

Suggestion 2: explore designs for explicit and visual task coordination, to counter-act hidden [11] implicit coordination.

Reflection 3: **Levels of Attention** When reflecting on process-responsibility (**Finding 4**) and tool-information (**Finding 2**), we note that developers at different levels of experience take on different roles in the review, and in doing so they look for different things. We hypothesise that rather than it being the case that more experienced developers always look at a higher level, it is their expertise that allows them to look at different levels at once, or switch rapidly between them. However, they do this largely without tool support, and often in a way that causes friction with the single, fixed, implicit unit of analysis (see above).

Suggestion 3: explore tool designs supporting different levels of attention and support for switching between these.

Reflection 4: **Level of Control** As a reflection connecting to tool-size (**Finding 3**) and tool-information (**Finding 2**), we note that the developer's ability to manipulate their change sets to create the right unit of analysis is an important aspect of the review task. However, we have seen in previous work [7] that this is by no-means easy, version control systems can be hard to manipulate and developers often fall back to adopted ritual behaviours; they are so glad when they have got something, anything, into version control effectively that shaping it as a social space is not their priority. The effective result of this is that the unit of analysis is often left essentially determined by chance and the path the developer took, interfering with the efficiency of the review process, causing delays and cost.

Suggestion 4: explore designs for increased level of control of the unit of analysis.

Summary. In well established, well resourced, centralised practices, the inflexibility of review tools and practices may help enforce central authority and provide mobility for people already familiar with the practices. However, we know that software engineering contexts vary greatly in what would be the most effective mode for a given circumstance. In these contexts, the implicit nature of the unit of analysis, and the lack of tools for explicit co-ordination, between reviews or of management of attention within them may well hinder rather than the help the goal of building better software and more effective teams. We see a possible connection between the challenge that 'one size does not fit all' and the superlativism hypothesis rebutted by Green et al. [12]. Outlining that rather than designing a solution assumed to be the best, it is necessary to study the alignment between different models of understanding and action of the involved stakeholders. This may offer a productive theoretical approach to further developing this research in the future.

6 THREATS TO VALIDITY

Internal threats. In the interview situation, we represented the code review process which may have caused a response bias, where participants limited their critique of the practice of code review. We believe we counteracted this effect in the study by the use of triangulation and the follow-up survey, where we saw more critique of the process and respondents admitting to the process being monotone and boring. In the survey design, we introduced the tool-task

misalignment concept and then asked respondents about this phenomenon, and in doing so biasing the follow-up questions towards our description of tool-task misalignment. To limit this effect we asked about obstacles and challenges (Q5) before introducing the concept of tool-task misalignment (making it appear first on the next page of the survey). We also included a question where we asked respondents to describe a typical tool-task misalignment (Q6). There is also a risk of a selection bias in how we selected interview participants, where we selected a contact person and the contact person carried out the selection by asking colleagues. Furthermore, we cannot exclude that some of the interviewees also answered the survey. We have tried to ask interviewees to refuse from participating in the survey but could not get many confirmations from the interviewees. If all 12 interviewees had participated in the survey, their answers could amount to 15–27% of the survey answers. This could lead to some bias. However, we believe that few, if any, of the interviewees did participate and that the survey results show similar trends largely independently from the interview results. Since the response rate for the survey was low, there might be a risk for self-selection bias.

External threats. We carried out our study at two multi-national companies. From the data we gathered, the companies are using a common setup for code review with tools like Gerrit, GitHub, and GitLab. We believe this common code review setup makes our results more generalizeable, but at the same time our study is initial and exploratory with a limited number of participants in both the interview study and in the survey. From the data we gathered, the code review process at the selected companies appear to be mature and well-established (Table 3), which may limit the extent to which what we found would transfer to a context were code review is new. The subjects in our study were fairly experienced in code review (Table 2), which may limit to what extent our results would transfer to a group of subjects with less code review experience. We further note that the subjects in our study report on use of mainly three review tools (Gerrit, GitHub, GitLab), with a bias towards Gerrit, which may limit to what extent the results can be transferred to other tools. We believe the broad use of the tools mentioned in our study helps to reduce this risk.

7 CONCLUSIONS

Driven by a wish to further understand the developer's experience of the code review process we have carried out an initial exploratory mixed-method study with interviews followed by a survey, with subjects from two multi-national companies. In analysing the gathered data, we have characterized what we have seen in terms of misalignments between the reviewer, the tool, and the process. We have focused on the phenomenon of tool-task misalignment, where developers leave the code review tool (e.g., due to lack of information), and have found this to be a broader phenomenon.

Based on our results, we have provided reflections paired with suggestions for future exploration of interventions into the code review process via tools and processes. These suggestions provide several opportunities for future explorations and provide fertile ground for an extended study beyond our initial exploration presented in this paper.

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] A. Bacchelli and C. Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 712–721.
[2] D. Badampudi, R. Britto, and M. Unterkalmsteiner. 2019. Modern code reviews – preliminary results of a systematic mapping study. In *Proceedings of the Evaluation and Assessment on Software Engineering (EASE)*. 340–345.
[3] T. Baum, H. Leßmann, and K. Schneider. 2017. The choice of code review process: A survey on the state of the practice. In *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES)*. 111–127.
[4] T. Baum, O. Liskin, K. Niklas, and K. Schneider. 2016. Factors Influencing Code Review Processes in Industry. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. 85–96.
[5] V. Braun and V. Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
[6] 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 International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 531–535.
[7] L. Church, E. Söderberg, and E. Elanga. 2014. A case of computational thinking: The subtle effect of hidden dependencies on the user experience of version control. In *PPIG'14*.
[8] M. Ciolkowski, O. Laitenberger, and S. Biffl. 2003. Software reviews, the state of the practice. *IEEE Software* 20, 6 (2003), 46–51.
[9] N. Davila and I. Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software* 177 (2021), 110951.
[10] 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.
[11] T.R.G. Green. 1989. Cognitive Dimensions of Notations. *People and Computers V.* (1989), 443–460.
[12] Thomas Green, Marian Petre, and Rachel Bellamy. 1991. Comprehensibility of visual and textual programs: A test of superlativism against the 'match-mismatch' conjecture. *Empirical Studies of Programmers: Fourth Workshop* (01 1991).
[13] Persson J. and Rydenfält C. 2021. Why Are Digital Health Care Systems Still Poorly Designed, and Why Is Health Care Practice Not Asking for More? Three Paths Toward a Sustainable Digital Work Environment. *Journal of Medical Internet Research* 23 (2021). Issue 6.
[14] O. Kononenko, O. Baysal, and M. W. Godfrey. 2016. Code Review Quality: How Developers See It. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1028–1038.
[15] 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.
[16] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.
[17] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proc. ACM Hum.-Comput. Interact.* 2, Article 135 (2018).
[18] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli. 2018. Modern Code Review: a Case Study at Google. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 181–190.
[19] J. Shimagaki, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi. 2016. A study of the quality – Impacting practices of modern code review at Sony Mobile. In *Proceedings of the International Conference on Software Engineering Companion (ICSE Companion)*. 212–221.
[20] SmartBear. 2020. *The 2020 State of Code Review 2020 Report.* Technical Report. SmartBear. https://smartbear.com/resources/ebooks/the-state-of-code-review-2020-report.
[21] 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 International Conference on Software Engineering (ICSE)*. 677–687.
[22] E. Söderberg, L. Church, J. Börstler, D. C. Niehorster, and C. Rydenfält. 2022. What's bothering developers in code review?. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.
[23] T. Winters, T. Manshreck, and H. Wright. 2020. *Software Engineering at Google: Lessons Learned from Programming over Time.* O'Reilly Media.