Witter dos Santos and Nunes *Journal of Software Engineering Research and Development* (2018) 6:14
https://doi.org/10.1186/s40411-018-0058-0

Journal of Software Engineering Research and Development

RESEARCH Open Access

Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data

Eduardo Witter dos Santos* and Ingrid Nunes

*Correspondence:
eduardo.witter@ufrgs.br
Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

Abstract

Code review is a potential means of improving software quality. To be effective, it depends on different factors, and many have been investigated in the literature to identify the scenarios in which it adds quality to the final code. However, factors associated with distributed software development, which is becoming increasingly common, have been little explored. Geographic distance can impose additional challenges to the reviewing process. We thus in this paper present the results of a mixed-method study of the effectiveness of code review in distributed software development. We investigate factors that can potentially influence the outcomes of peer code review. The study involved an analysis of objective data collected from a software project involving 201 members and a survey with 50 practitioners with experience in code review. Our analysis of objective data led to the conclusion that a high number of changed lines of code tends to increase the review duration with a reduced number of messages, while the number of involved teams, locations, and participant reviewers generally improve reviewer contributions, but with a severe penalty to the duration. These results are consistent with those obtained in the survey regarding the influence of factors over duration and participation. However, participants' opinion about the impact on contributions diverges from results obtained from historical data, mainly with respect to distribution.

Keywords: Code review, Distributed software development, Empirical study, Survey

1 Background

Code review is a common practice adopted in software development to improve software quality based on static code analysis by peers. There are studies that provide evidence that it reduces the number of defects detected after release, mainly when it has adequate code coverage as well as engagement and participation of reviewers (McIntosh et al. 2014). Moreover, code review is a recognized way to foster knowledge sharing that benefits authors and reviewers (Hundhausen et al. 2013). It also improves team collaboration because it creates collective ownership of the source code, which results from collaborative work rather than individual work (Bacchelli and Bird 2013; Thongtanunam et al. 2016b). Nowadays, code reviews are less formal than in earlier decades of software development. In the past, it was typically in the form of code inspections (Fagan 1986), which

required formal meetings and checklists (Kollanus and Koskinen 2009). Today, such a practice is more informal, being referred to as *Modern Code Review* (MCR) (Bacchelli and Bird 2013). It is often assisted and enforced by tools, such as Gerrit (Google 2017a).

The effectiveness of code review depends on different factors and, when it cannot provide expected benefits, it becomes a costly and time-consuming task (Czerwonka et al. 2015; Thongtanunam et al. 2016a). For example, if there is a time gap between the completion of a change and its review by a peer, the author may have its work partially blocked, possibly affecting the whole software release (Thongtanunam et al. 2015b). This lack of dynamism in the code review activity increases the work in progress of teams, as new tasks are started while waiting for the pending reviews. Furthermore, the context switching between coding tasks and reviews may also have a negative impact on developers' work.

To understand the factors that positively and negatively affect the effectiveness of code review, previous studies were performed, e.g. (Thongtanunam et al. 2015a; Baysal et al. 2016; Yang 2014; Bosu et al. 2015). Examples of investigated factors are the patch size, the nature of the change, or author's company—that is, both technical and non-technical factors have been investigated. Moreover, to evaluate effectiveness, different criteria have been adopted, such as the review duration and the number of defects found after code review. As a result, relevant conclusions regarding code review have been reached. For instance, developers from other teams provide fewer but more useful feedback than those from the same team (Bosu et al. 2015). Despite all the significant results obtained so far, code review has been investigated only to a limited extent in the context of *geographically distributed software development* (Sengupta et al. 2006), which is becoming increasingly common over the last decades. In the late 90s, researchers focused on enabling formal code inspections, which involve meetings, in distributed scenarios (Perpich et al. 1997; Stein et al. 1997). In modern code review, in contrast, tool support and asynchronous communication help deal with geographic distribution. However, the effects of geographic distribution on the outcomes of code review (such as duration or reviewer engagement) have not been explored. Recent studies of code review in distributed software development are limited to experience reports on code inspection (Meyer 2008).

We thus in this paper focus on exploring how both technical and non-technical factors influence a set of metrics that are indicators of the effectiveness of code review in the context of Distributed Software Development (DSD). We present the results of a mixed-method study in which we investigated the relationship between four influence factors—namely number of changed lines of code, involved teams, involved locations and active reviewers—and the effectiveness of code review. As there is no single objective metric that captures whether a review is effective, we measured and analyzed different review outcomes that can be seen as an indication of the review effectiveness, such as reviewer participation and number of comments. The study involved (1) an analysis of objective data collected from a software project; and (2) a survey with 50 practitioners with experience in code review. This study is an extension of our previously presented work (Witter dos Santos and Nunes 2017), which was complemented by the survey that allows us to compare the results obtained with both research methods.

The first part of our study, referred to as *repository mining*, is based on a large amount of data (8329 commits and 39,237 comments) extracted from the code review database

of a project with 201 members during 72 weeks. The analysis of our results allowed us to conclude that a high number of changed lines of code tends to increase the duration of the review process with a reduced number of messages, while the number of involved teams, locations and participant reviewers generally improve the contributions from reviewers, but with a severe penalty to the duration. These results are consistent with those obtained in the survey regarding the influence of factors over duration and participation. However, participants' opinion about the impact on contributions diverges from results obtained from historical data, mainly with respect to distribution.

The remainder of this paper is organized as follows. We first discuss related work in Section 2. We then provide details of our target project in Section 3, describing the code review process of our target project. Next, we describe our study settings in Section 4. The results of the first and second parts of our study are presented and analyzed in Section 5. A discussion regarding obtained results is presented in Section 6, followed by our conclusions, which are presented in Section 7.

2 Related work

Since the pioneering work of Fagan (1976) on formal code inspections, many researchers proposed approaches to improve this well-structured and phased form of code review (Parnas and Weiss 1985; Bisant and Lyle 1989; Martin and Tsai 1990). With the popularity of DSD, other researchers investigated how to make code inspections feasible when the involved people cannot physically meet in a particular location (Perpich et al. 1997; Stein et al. 1997). Despite its popularity among researchers and practitioners, formal code inspection and its variations have received less attention since the early 2000s (Kollanus and Koskinen 2009).

More recently, much work focusing on *modern* code review has been done, ranging from studies that investigate what leads to successful code review to approaches that recommend suitable reviewers. For example, in Balachandran (2013)'s approach, recommended reviewers are those that made the most recent changes in the portion of code to be reviewed. His approach was improved by Thongtanunam et al. (2014), for projects with specific characteristics, using the File Path Similarity (FPS), which takes into account previous changes with similar paths or file names. These approaches were extended by also considering similarity among past commit messages (Xia et al. 2015) and recent activity of the possible reviewers (Zanjani et al. 2016). Viviani and Murphy (2016) took another direction by prioritizing pending reviews for each reviewer instead of finding the best candidate reviewers for a given change. This is motivated by the fact that several projects have a high concentration of review requests in a small group of contributors (Yang 2014).

Despite all these significant contributions to the field of code review, it is crucial to understand the factors that influence the effectiveness of code review to, for example, provide foundations to improvements while making reviewer recommendations. Therefore, many studies focus on providing a deeper understanding of code review, and its influence factors (e.g. number of changed lines of code and experience of individuals) and outcomes (e.g. duration and discussion among reviewers). Although such studies are similar to ours, they do not focus on DSD. We next discuss technical and non-technical influence factors investigated in existing studies.

Investigation of technical factors Different correlations were studied involving technical factors. Thongtanunam et al. (2015a)'s study provided evidence that reviewers are less rigorous and find fewer defects on files with a high incidence of defects in the past, focusing on superficial aspects, such as coding standards rather than on functional aspects. In a more recent study (Thongtanunam et al. 2016a), the same authors identified that bug fixes typically receive the first feedback faster than implementations of new features. Moreover, they reported that changes with detailed and explanatory commit messages have lower stale rates, while those that are poorly described receive less attention of reviewers.

Focusing on the code review duration (in working days), a few influence factors were investigated. Bosu et al. (2015) concluded that the patch size affects the duration in most of the analyzed cases, while task priority in the release plan and the affected software components have only occasionally influenced some of the projects analyzed by Baysal et al. (2016).

Investigation of non-technical factors Non-technical factors also received attention recently. As stated by Czerwonka et al. (2015), the social network that naturally emerges inside the companies or projects should be considered as well as the specific reviewers' skills and their availability and willingness to review. An analysis of the social network of three open source projects (Yang 2014) revealed that the most active reviewers have central roles in the social network of those projects and are frequently some of the most important contributors. Bosu et al. (2015) observed, in a particular organization, that 75% of the code review feedbacks come from members of the author's team, but are slightly less useful than those from other teams. Baysal et al. (2016), in turn, pointed out that when multiple organizations contribute to the same project, the code review can take more time to be completed and have higher rejection rates depending on which organization is authoring or reviewing a patch, based on the analysis of several case studies.

The experience of authors of the code under review has also been pointed out as relevant in code review. Senior members of the company and those with recognized expertise usually receive more priority, faster and more detailed feedback, enabling a faster code review with better results for the quality (Baysal et al. 2016; Rahman et al. 2016). The experience of the reviewers is relevant as well, based on results of the investigation of a large company (Bosu et al. 2015)—the quality of provided feedback increased during the first year in the company and then stabilized in a *plateau*.

In a study involving three large open source projects, Thongtanunam et al. (2016a) also investigated non-technical factors, focusing on how the code review was affected by prior events on the files under review. Their conclusions are (1) files that received a slow initial feedback in the past will also likely receive slow feedback in the future; (2) files with more authors and reviewers in the past receive more attention; and (3) the number of changed files, directories and the length of the commit message are also important.

Summary Given that many factors that influence code review have been investigated, we summarize what each previous study analyzed in Table 1. Rows in this table consist of the examined influence factors, while columns represent the analyzed outcomes associated with code review. In cells, we list the studies that focused on the relationship between a given influence factor and outcome.

Table 1 Influence factors and review outcomes investigated by previous studies

Influence factors	Absence of discussion	Comment usefulness	Feedback delay	Participation in review	Post-release defects	Review duration	Review iterations	Review quality
Amount of previous defects	♣		♣	⋆				⋆
Affected modules						♠		
Author experience				♣		♠		
Author's company						♠		
Bug fix or new feature			♣				△	
Commit message size	♣							
External reviewers		◇						
Length of prior discussions	♣				♡			
Number of authors	♣		♣					
Number of reviewers	♣							
Patch size (Files)	♣	◇						
Patch size (LOC)						♠	△	
Prior feedback delay			♣					
Priority						♠		
Review coverage					♡ †			
Review speed					♡			
Reviewer experience		◇				♠		
Source code type		◇						

♣ Thongtanunam et al. (2016a)
◇ Bosu et al. (2015)
♡ McIntosh et al. (2014)
♠ Baysal et al. (2016)
△ Beller et al. (2014)
⋆ Thongtanunam et al. (2015a)
† Shimagaki et al. (2016)

Some of these studies analyzed MCR targeting FLOSS (Free, Libre and Open Source Software) projects, such as OpenStack, Qt, and LibreOffice, which present DSD characteristics. However, we emphasize that most of these studies did not investigate the impact of distribution: factors associated with distribution were random variables rather than independent variables. For instance, Baysal et al. (2016) reported that some analyzed companies had co-located groups, while others used DSD, without treating this issue as a dependent variable. Similarly, Bosu et al. (2015) found that comments from other teams are slightly more useful, but without considering co-location or the number of involved teams.

As can be seen, different combinations of influence factor and outcome have been analyzed. Differently from previous work, our study focuses on DSD and, therefore, we focus on other influence factors, such as the number of involved cities and teams. Some of our investigated factors, e.g. patch size (LOC), have already been studied, but not in a DSD scenario. Moreover, we analyze four different outcomes of code review, which are described in next section together with other details of our study settings.

3 Study subject

Our study is based on the analysis of data collected from a (commercial) software project and developers from a single software development company. Due to the project size, we were able to collect a large amount of information regarding its code review. We next describe the code review process of the project, provide details about the collected data, and characterize the participants of our survey. No further information can be given due to a confidentiality agreement.

3.1 Code review process

We overview the code review process followed in the target project in Fig. 1. First, authors send a piece of code to be reviewed. Anyone can, at any point in time, invite reviewers or add itself as a reviewer, what would allow any (interested) developer to contribute. Moreover, in our target project, Gerrit is configured with the *reviewers-by-blame* plugin (Google 2017c), which automatically adds reviewers based on the last changes made on the files to be reviewed, as proposed by Balachandran (2013). The code is then analyzed by automated reviewers that check several quality criteria, such as compilation, cyclomatic complexity, lack of documentation, failed unit tests, among other static analysis and

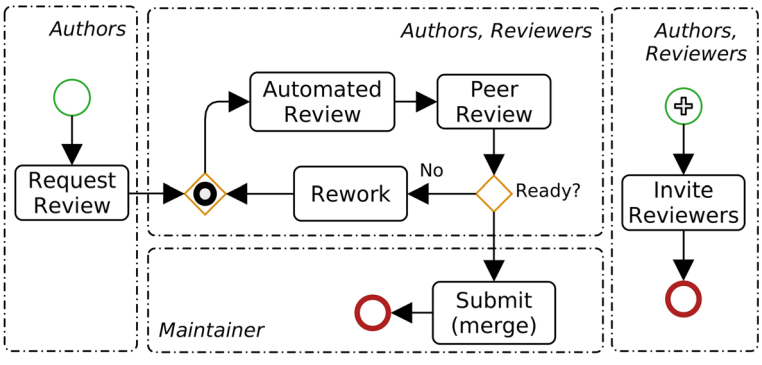

Fig. 1 Overview of the Code Review Process

runtime verifications. This automated verification usually takes less than 15 min to execute and rejects the change if any critical test fails, so that the author can fix the reported issues. Human reviewers and authors can discuss, ask and provide suggestions for each line of code. Moreover, each reviewer can vote to summarize its feedback using one of the following values.

Veto The reviewer considers that the change cannot be integrated without fixing the reported issues or answering questions made. This prevents the commit to be merged.

Rejection The reviewer *recommends* fixes before the change is merged.

Neutral The reviewer typically *asks* easy questions to be answered.

Acceptance The reviewer considers that, though the change it adequate, it needs more reviews from other developers.

Approval Only maintainers of the module associated with the commit have this kind of vote, as they are responsible for the module quality. Maintainers can perform technical reviews, but must also verify that relevant developers are not missing in the list of invited reviewers and that the overall state of the code review is adequate.

It is important to note that all invited reviewers, but the maintainer, are not obliged to provide feedback. Before approving the change, the maintainer of each module should consider if the most important reviewers already reviewed the code. In the end, the piece of reviewed code is submittable if all the following conditions are satisfied: (i) there is no rejection from automated reviewers; (ii) there is no veto; and (iii) the maintainer has approved the change. If all these conditions hold, the maintainer is able to merge the change into the destination branch.

3.2 Analyzed data

The target project of this study involves the development of an operating system for embedded systems of routers and switches, using the C, C++, and Yang (Internet Engineering Task Force (IETF) 2017) languages. This project has a total of 269 repositories, from which 63 are dedicated to test automation, using Python, Vagrant, and Ansible. We consider that the operating system code and its tests are part of the same project, as the developers implement both firmware and tests for each task. All repositories are configured to reject the merges without code review.

The mined data refers to a period of 72 weeks, starting in October 2014. In the collected data, we had a total of 11,109 code reviews. After filtering these data (see next section), we obtained 8329 code reviews associated with 39,237 comments (an average of 4.7 comments by review). Such code reviews are associated with: (i) 201 experienced developers; (ii) 4 development locations in 4 different cities in the same country and time zone; and (iii) 21 different teams. Members of a given team work on the same location, i.e. there are no distributed teams. All teams are organized as *feature teams* and use Scrum with three-week sprints to release new software versions every three months. A continuous integration pipeline is used to run functional tests on several test environments that contain network topologies with real products and emulators.

3.3 Survey participants

Our goal by conducting a survey with software developers of the same company is to be able to compare the perceptions of developers with concrete objective data. The survey was conducted in January 2018, when we randomly invited 80 developers to participate. From them, 50 voluntarily provided a response within a week (response rate of 62.5%). However, 5 participants reported (very) low experience in code review (as author or reviewer) and these were discarded because they could provide unreliable answers. Because our survey involved other projects and a different time period of the first part of the study, many participants were not developers of our target project during the period in which we collected data. However, the projects of which participants are members use Gerrit to implement and reinforce modern code review practices. Although there may be divergences in the software development process as a whole, all these projects adopt the code review workflow described in Fig. 1.

Table 2 provides detailed information about 45 participants, including age, education and experience. Answers that indicate the participant experience—used solely for the purpose of characterizing our sample—were self-reported based on the participant's subjective view. We can see that 97.8% of the participants reported medium to very high experience with projects with multiple teams, whereas 93.3% reported medium to very high experience with projects with multiple locations, suggesting that they have experience in DSD.

4 Study settings

After discussing our target project, we now proceed to detailing our study. We first state our goal and research questions, then describe collected metrics and finally the procedure of the two parts of our study, namely repository mining and survey.

4.1 Goal and research questions

To design our study, we followed the goal-question-metric (GQM) paradigm (Basili et al. 1986). Therefore, we first specify our goal using the GQM template and derived research questions. Our goal is detailed next.

To understand the factors that influence code review in distributed software development, characterize and evaluate the relationship between different influence factors and code review effectiveness from the perspective of the researcher as code review is performed by software developers in a single project study.

Based on this goal, we derived a set of research questions, each associated with one of the influence factors investigated in our study. There are both technical and non-technical factors. As said, although some have been investigated in the past, it is our goal to analyze them in the context of DSD. For short, we refer to our investigated scenario as *distributed code review*. Our research questions are listed as follows.

RQ-1: Does the number *lines of code to be reviewed* influence the effectiveness of distributed code review?

RQ-2: Does the number of involved *teams* influence the effectiveness of distributed code review?

RQ-3: Does the number of involved *development locations* influence the effectiveness of distributed code review?

Table 2 Demographic data of survey participants (N = 45)

Characteristic	Answer	#	%
Age	20-29 years	11.0	20.0
	30-39 years	25.0	54.6
	> 39 years	9.0	24.4
Gender	Male	44.0	97.8
	Female	0.0	0.0
	Not informed	1.0	2.2
Education	Graduation	35.0	77.8
	Master (incomplete)	2.0	4.8
	Master	7.0	15.6
	PhD	1.0	2.2
Professional experience in software development	2-5 years	4.0	8.9
	5-10 years	17.0	37.8
	> 10 years	24.0	53.3
Experience as reviewer	Very low (1)	-	-
	Low (2)	-	-
	Medium (3)	14.0	31.1
	High (4)	26.0	57.8
	Very high (5)	5.0	11.1
Experience as author	Very low (1)	-	-
	Low (2)	-	-
	Medium (3)	13.0	28.9
	High (4)	28.0	62.2
	Very high (5)	4.0	8.9
Experience in projects with multiple teams	Very low (1)	0.0	0.0
	Low (2)	1.0	2.2
	Medium (3)	12.0	26.7
	High (4)	22.0	48.9
	Very high (5)	10.0	22.2
Experience in projects with multiple locations	Very low (1)	0.0	0.0
	Low (2)	3.0	6.7
	Medium (3)	12.0	26.7
	High (4)	21.0	46.7
	Very high (5)	9.0	20.0

Participants that reported (very) low experience as author or reviewer were discarded

RQ-4: Does the number of *active reviewers* influence the effectiveness of distributed code review?

We investigate the number of teams and locations separately because the former captures distribution among teams, allowing us to analyze the impact of involving reviewers that have different project priorities and goals (possibly conflicting) and limited interaction, while the latter additionally captures the impact of geographic distribution.

4.2 Influence factors and outcomes

Each research question is associated with an influence factor to be investigated, with respect to their impact on the effectiveness of distributed code review. However, there is no unique metric to measure review effectiveness. Therefore, we consider a set of outcomes of code review, which are measured. They can be used as indicators of the

review effectiveness. Before detailing these outcomes, we next further specify our influence factors—which are listed following the order of our research questions—detailing how they are measured.

Patch Size (LOC) The patch size (LOC) is used to refer to the number of lines of code added or modified in a commit and thus need to be reviewed. This lines of code considered are those present in the final version of the code, after going through the reviewing process.

Teams Teams refer to the number of distinct teams associated with the author and invited reviewers. If the author and all reviewers belong to the same team, the value associated with this influence factor is 1.

Locations Locations refer to the number of distinct geographically distributed development sites associated with the author and invited reviewers. If the author and all reviewers work in the same development site, the value associated with this influence factor is 1.

Active Reviewers Actives reviewers are those that actually participate in the reviewing process—with comments or votes—from those invited. Although this can be seen as an outcome of the review, given that there is no control of how many of the invited reviewers will actually participate, we aim to explore if the number of active reviewers influence other outcomes, such as duration. Therefore, active reviewers are investigated as an influence factor, consisting of the number of reviewers that contributed to the review.

Now we focus on describing the analyzed code review outcomes that indicate the review effectiveness. Code review is effective when it achieves its goals, which can be untimely to identify defects in the code, issues related with code maintainability and legibility, or even to disseminate knowledge. However, these goals might include constraints regarding the impact in the development process and invested effort.

It is not trivial to evaluated whether these goals are achieved. For example, Bosu et al. (2015) created a model to evaluate whether the comments of a code review are useful based on the text of the given comments. This measurement, however, may not be precise. In our work, we focus on measurements that are more objective.

We thus selected four objective outcomes, described as follows. The first is related to project time constraints, while the remaining three are related to the input from other developers (reviewers) leading to possibly less failures, code quality improvement and knowledge dissemination.

Duration (DUR) Duration counts how many days the code review process lasted, from the day that the source code is available to be reviewed to the day that it received the last approval of a reviewer.

Participation (PART) Participation consists of the fraction of invited reviewers that are active, ranging from 0% (no invited reviewer participates) to 100% (all invited reviewers participate). Automated reviewers are not taken into account.

Comment Density (CD_G) Instead of simply counting the number of review comments, we take into account the amount of code to be reviewed. Therefore, comment

density refers to the number of review comments divided by the number of groups of 100 LOC under review, thus giving the average number of review comments for each 100 LOC. Review comments can be any form of interaction, e.g. approval, rejection, question, idea or other types of comments made by any reviewer—votes count as comments because they are a form of input and have a particular meaning. A multiplying factor of 100 is used to avoid small fractioned numbers, which are harder to compare and less intuitive. Comments from automated reviewers are ignored, as this type of feedback is a constant, regardless of human interactions.

Comment Density by Reviewer (CD_R) It is expected that the higher the number of reviewers or teams, the higher the number of comments. Therefore, CD_G alone can lead to the wrong conclusion that discussions were productive when many reviewers are involved. We thus also analyze comment density by reviewer, given by the division of the comment density by the number of active reviews (without taking into account automated reviewers).

As we are interested in the effectiveness of code review, we next describe what we consider an effective code review based on the outcomes considered in the study.

Too short or too long code review. There are studies (Kemerer and Paulk 2009; Ferreira et al. 2010) that suggest time constraints for code review activities, limitation on the number of lines reviewed per hour and also the total amount of hours spent doing code review in a single day. Such limitations are imposed because the code review may become error-prone or even consume more time and resources to be finished due to tiredness. Moreover, if the review takes too long (i.e. high duration) to be completed, developers may be prevented to continue their work and also work does not get done. Therefore, shorter code reviews are preferred. However, if such review is too short, it may also mean that reviewers have not properly analyzed the change.

Low reviewer participation. When reviewers are invited to participate in the review, it is expected that they contribute. However, not all participate. Therefore, the higher the participation of reviewers, the better. Nevertheless, we do not expect that participation is 100%, given that there are developers that are invited automatically and may not be relevant reviewers anymore.

Few contributions from reviewers. Reviewers may contribute in different ways, ranging from a simple vote to long discussions. We assume that the higher the number of comments made by reviewers, the more fruitful the discussion and consequently the more effective the review. However, as explained, we do not consider the absolute number of comments, but its density considering the amount of code to be reviewed. Moreover, we consider the amount of contribution generally (CR_G) and by reviewer (CR_R). For both, the higher, the better.

Although in some situations a low number of comments (either generally or by reviewer) is enough—for example, when a low number of comments helped to improve the code, or the change to be reviewed is minor—note that these might be not the usual case. Because we analyze a high number of code reviews, these exceptional cases do not

significantly impact the results. Moreover, votes count as comments; consequently, even if there is no need for long discussions, it is important to have at least the acknowledgement of the reviewer in the form of a vote, i.e. a comment. Finally, we also analyze duration and participation, which complement the analysis of the effectiveness of code review.

4.3 Procedure: repository mining

In short, the procedure of the first part of our study consists of the extraction of information from a code review database and analyses of the relationship between influence factors and outcomes. In this section, we detail how we operationalized this.

Our data is extracted from Gerrit[1], a tool that provides the management of Git repositories with fine-grained control over the permissions for users and groups. It also provides a mechanism to implement code review, with its associated approvals, allowing votes, comments, and edition of the source code. Every interaction among authors and reviewers is recorded, including the comments and votes of the *review bots*, which are automated reviews. It also provides a sophisticated query mechanism to get information about all open and closed reviews. We next describe the steps taken to obtain, process and filter the data for this study using Gerrit.

Getting raw data from the code review database First, we fixed a time frame in the past so we can get data completed code reviews. Gerrit provides a query mechanism (Google 2017b) that can be used to get structured information about code reviews in JSON format. One query for each week had to be made due to the limitation of obtaining at most 500 results per query.

Parsing and filtering code review information The retrieved JSON files provided part of our required data. The remaining data had to be computed from the raw data obtained from the internal Gerrit database model. The resulting data was filtered, discarding some reviews of some types of modules, described next.

1 *Documentation.* Some repositories are used exclusively for internal documentation of the project (processes and products), and have a different workflow and time constraints.
2 *Third-party software.* Some repositories are maintained by open source communities or component vendors. Local internal copies of these repositories exist due to traceability and to avoid downloading them multiple times. Reviewers do not review code in these repositories.
3 *Binary artifacts.* Some repositories contain binary files, e.g. images and libraries. These files are not reviewed and, if considered, would (incorrectly) increase the patch size.

Representing and analyzing data Given that we have four research questions with four associated influence factors as well as four outcomes, there is a large amount of data to be analyzed. Our data consists essentially of continuous or discrete positive numbers, with different scales and ranges. For example, there are only four involved locations while the patch size can be up to approximately 4 KLOC. To deal with these discrepancies, we adopted an approach similar to that of Baysal et al. (2016). We clustered data in

groups, representing the variance of outcomes in each group using box plots. Additionally, we performed statistical tests to identify groups that are significantly different from each other.

4.4 Procedure: survey

Our survey collected anonymous data from participants. The questionnaire they were given includes four main parts: (i) presentation of the study and consent to participate; (ii) demographic data of participants; (iii) participant background and experience; and (iv) questions about how outcomes of code review are affected by influence factors.

In the first part of the questionnaire, we briefly introduced modern code review and stated the goal of our survey. We explicitly declared that the participation was voluntary and informed participants that any information that could reveal the their identity would be kept confidential. Next, we presented the adopted terminology to ensure that terms used in the questions would be correctly understood. The introduced terms are: teams, locations, authors, reviewers and active reviewers. Next, participants were asked about their demographic information as well as background and expertise, as shown in Table 2.

Finally, we asked participants to assess how they evaluate the relationship between influence factors and review outcomes. The considered influence factors are the same as above. However, as outcomes, we considered duration, participant and number of comments. The reason to not distinguish total comments and comments by reviewer is due to our assumption that it is hard for a participant, without data, to assess these separately. Questions associated with each outcome followed the prototype presented in Table 3, replacing X by the name of the outcome.

For each influence factor, we used a 5-point Likert scale, so participants could provide one of six possible answers: (1) much worse, (2) worse, (3) no influence, (4) better, (5) much better, or "I am unable to inform." Our questions assumed that there is a directly or inversely proportional relationship between influence factors and outcomes. If participants believe it was not the case, they were asked to state that they were unable to inform and describe their opinion in an open-ended question. Our questionnaire was validated with a pilot study.

5 Results

Having described our study settings, we proceed to the presentation of obtained results. They are presented according to our research questions, and in each of them, we discuss results associated with each of our investigated outcomes.

5.1 Repository mining

In this section, we present the results of the part of our study that is based on objective data, which was obtained by mining the software repositories of the analyzed project.

Table 3 Prototype of the main questions of the survey

For higher values of	Outcome X is...				
	Much worse	Worse	No influence	Better	Much better
Patch size (LOC)					
Teams					
Locations					
Active reviewers					

5.2 RQ-1: patch size (LOC)

The first influence factor we analyze is the patch size in terms of LOC. We split our data into 13 groups according to this factor, listed in the first column of Table 4. This table shows our obtained results regarding this influence factor—mean (M) and standard deviation (SD)—for each group, considering each review outcome. We also show the number of code reviews in each group as well as values associated with all reviews, in order to be able to compare overall values with values of each group. For better visualizing the results, they are also presented in Fig. 2.

We analyze the influence of patch size in review outcomes by comparing the means across the different groups. Given that our data does not have a normal distribution, we use the Kruskal-Wallis test (nonparametric) to verify whether there is a statistically significant difference among the means. This is also the case for the other analyzed influence factors and outcomes. When there are significant differences, we use Dunn's test for post hoc tests, because the compared groups have different sizes.

Our results show that there are significant differences across the different groups ($H = 1761.32, p < 0.05$). More specifically, larger patches take longer to have their review completed—which is expected. Kemerer and Paulk (2009) and Ferreira et al. (2010) pointed out that there must be a limit of LOC reviewed per hour by a single reviewer and a maximum number of hours of code review per day, in order to achieve good coverage and final quality. However, the relationship between patch size and duration is nonlinear; a linear least-squares regression model produced an r-squared value of 2.9%, which means that a linear model has low explanatory power for the data. For example, the average time to review patches of 601–800 LOC is two times greater than the time to review patches of 61–80 LOC, which are ten times smaller. Among some groups, e.g. 21–40 LOC and 41–60 LOC, there is no statistically significant difference. Due to space restrictions, we do not report all significant differences among groups. They can be seen elsewhere together with more information of our results, such as medians[2].

Considering participation, we observed that the proportion of invited reviewers that actually provide feedback during the review process decreases when the patch size increases. The difference among the patch size groups is also statistically significant

Table 4 Outcomes by patch size (LOC)

LOC	#Rev	DUR		PART		CD_G		CD_R	
		M	SD	M	SD	M	SD	M	SD
0–20	3836	1.8	6.6	86.1	22.3	99.3	139.6	47.3	60.6
21–40	716	3.9	13.0	81.7	22.7	15.3	15.4	5.7	4.1
41–60	475	4.4	10.2	78.4	24.0	9.4	8.4	3.5	2.2
61–80	342	5.0	10.3	79.5	22.4	7.8	6.8	2.7	1.6
81–100	273	5.8	16.0	74.9	24.0	5.4	4.3	2.0	1.2
101–200	762	6.0	12.9	75.2	23.8	4.3	3.7	1.5	0.9
201–400	688	8.0	16.8	73.7	23.7	2.6	2.5	0.9	0.7
401–600	371	9.4	16.0	71.9	22.3	1.5	1.5	0.5	0.3
601–800	203	10.4	16.7	71.2	23.1	1.3	1.5	0.4	0.3
801–1000	158	10.0	15.7	69.4	24.6	1.0	0.9	0.3	0.2
1001–2000	282	13.4	19.1	66.9	23.2	0.8	0.8	0.2	0.2
2001–3000	78	15.1	18.9	62.0	24.7	0.4	0.3	0.1	0.1
> 3000	145	12.7	17.0	67.1	24.5	0.2	0.2	0.1	0.1
Total	8329	4.7	12.1	80.1	23.8	48.8	105.8	22.9	46.9

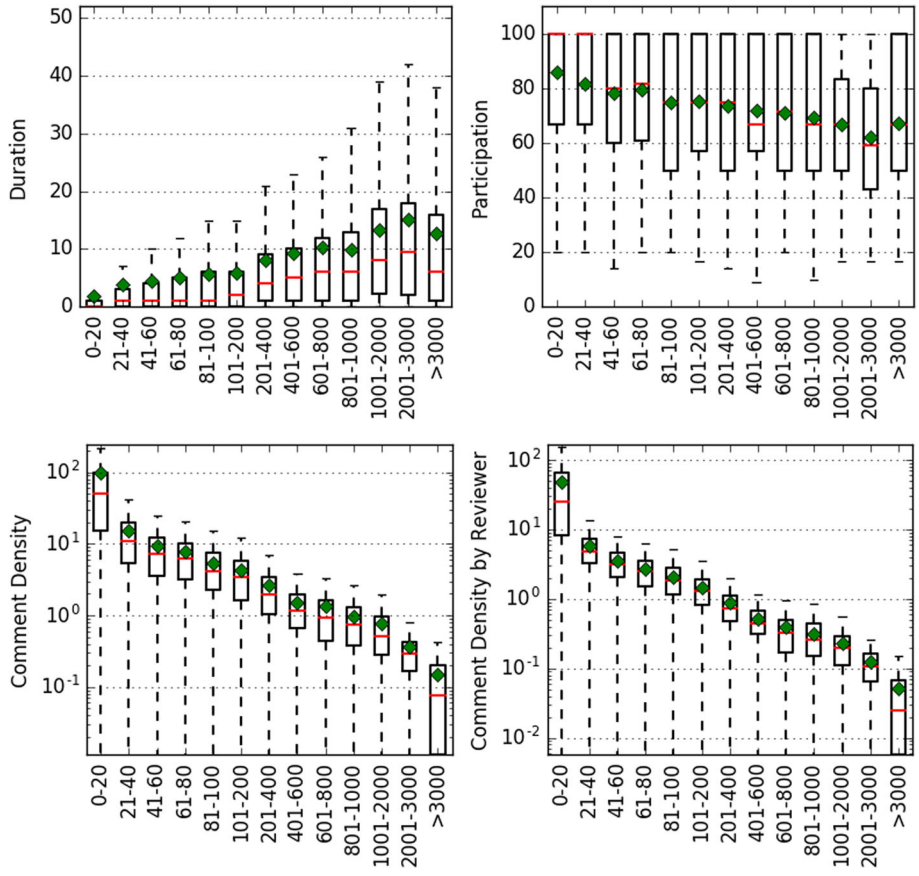

Fig. 2 Influence of the Patch Size (LOC) over Duration (days), Participation (%), Comment Density (comments per 100 LOC) and Comment Density by Reviewer (comments per 100 LOC per active reviewer). The y-axes representing Comment Density and Comment Density by Reviewer are in a logarithmic scale

($H = 709.58, p < 0.05$), mainly due to differences between groups with 600 LOC or less and larger groups. A possible explanation to this is that larger patches likely require more effort from reviewers, discouraging engagement in the process.

When reviewers participate in the code review, the amount of contribution is measured by the overall comment density and comment density per reviewer. Our data shows that in both cases the larger the patch, the lower the comment density. Regarding overall comment density, there are statistically significant differences ($H = 709.58, p < 0.05$). According to the post hoc tests, this is due only to smaller groups. There is a significant difference only among few groups with more than 601 LOC, but among groups with less LOC, there are significant differences in most cases. Similarly, the comment density by reviewer decreases as patches are larger ($H = 3579.57, p < 0.05$), showing similar results in post hoc tests. This indicates that the amount of contribution is highly affected as the patch size increases up to a certain point. Then, the amount of contribution is limited but does not decrease after the patch reaches a certain size (> 601 LOC in our study).

One possible explanation for the results regarding patch size is that the patch size has an intimidating effect on invited reviewers, because the time required to provide significant contributions increases. This invested time, in our target project, is not explicitly recorded and is not associated with deliverables considered more relevant, such as produced code.

Conclusions of RQ-1: The patch size negatively affects all outcomes of code review that we consider as an indication of effectiveness. Reviewers are less engaged and provide less feedback. Moreover, the duration is not linearly proportional to the patch size, which may affect the quality of code review.

As discussed in the related work section, other studies investigated the impact of the patch size in code review. Bosu et al. (2015) showed that for some projects the proportion of relevant comments decreased by 10%, when they compared changes in 40 files with changes in a single file, while Baysal et al. (2016) showed that changes with more LOC need more iterations to be concluded, but without considering the time interval. Each iteration is typically the result of an accepted feedback or comment. This indicates that results with respect to patch size in non-distributed scenarios also hold for our investigated scenario.

5.3 RQ-2: teams

Each code review has a list of involved people, authors, and reviewers, and each one works in a single team. Consequently, every code review has also a list of involved teams based on the list of involved people. The results regarding the number of involved teams *vs.* review outcomes are shown in Table 5 and Fig. 3. With respect to the groups with 6 and 7 involved teams, we have only two occurrences of code reviews associated with each of them. We thus omitted them from Fig. 3, for legibility.

According to our results, the duration of code review is considerably higher if more teams are involved, with higher mean and also standard deviation values. The latter means more dispersion, as can be seen in the corresponding box plots, having more durations that are outliers. This can be partially explained by the working dynamics of teams, which have different goals, tasks, and managers.

Furthermore, technical divergences are often extensively discussed. When only one team is involved, issues are addressed faster, usually mediated or decided by a senior team member or even by the team manager. However, when more teams are involved, the implicit hierarchy among reviewers becomes flattened and reaching a consensus becomes harder. In this case, divergences usually reach managers and are resolved after meetings, conferences or e-mail discussions, what slows down the review. In our results, there is a statistically significant difference across the different groups of numbers of involved teams ($H = 586.72, p < 0.05$). Comparing groups in a post hoc analysis, we observed that this is due to the differences among groups with five or less involved teams, except the difference between code reviews involving 4 and 5 teams.

Table 5 Outcomes by number of teams

Teams	#Rev	DUR		PART		CD_G		CD_R	
		M	SD	M	SD	M	SD	M	SD
1	4934	3.0	8.3	82.7	24.1	42.2	92.0	22.1	45.1
2	2440	5.8	13.5	76.8	24.0	55.1	112.1	25.5	51.1
3	766	9.4	19.3	75.5	20.1	66.4	140.5	22.1	46.3
4	152	12.4	19.0	73.5	17.7	77.3	177.8	17.8	39.7
5	33	24.1	39.1	73.5	15.8	50.5	79.8	8.9	14.0
6	2	21.5	20.5	63.0	8.4	8.1	7.1	1.6	1.4
7	2	19.0	16.0	58.5	21.5	5.2	0.9	0.6	0.0
Total	8329	4.7	12.1	80.1	23.8	48.8	105.8	22.9	46.9

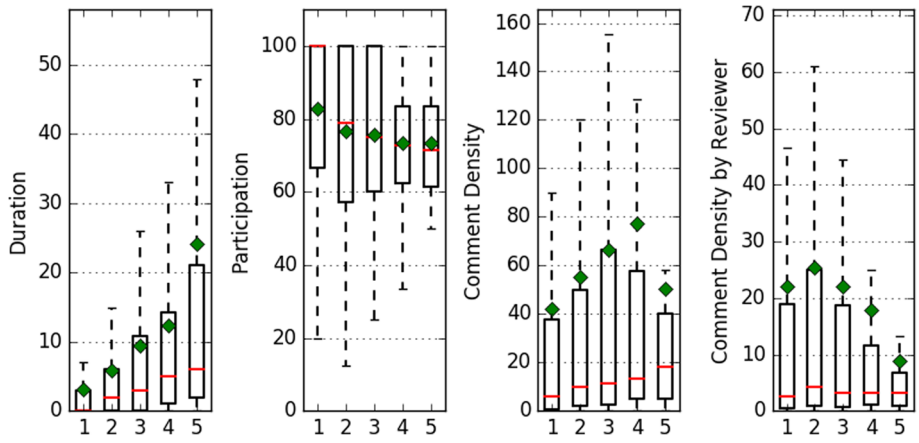

Fig. 3 Influence of the number of Teams over Duration (days), Participation (%), Comment Density (comments per 100 LOC) and Comment Density by Reviewer (comments per 100 LOC per active reviewer)

Similarly, there are also statistically relevant differences with respect to participation ($H = 226.72, p < 0.05$) and the post hoc analysis showed that the difference is only significant among reviews with four or fewer teams. However, the results indicate only a small negative influence on this review outcome.

Considering the effect on contributions, differences are also significant ($H = 184.71, p < 0.05$). Post hoc tests showed that this is due to the difference between reviews involving one team and the others. Although Fig. 3 indicates that the overall comment density increases together with the number of involved teams (except in the case of 5 involved teams), we can see in Table 5 that the standard deviation is high, indicating that results vary a lot, justifying the not significant differences. This can be explained by the specific teams involved, whether they are in the same location or not (an issue that is investigated in RQ-3). In our target project, there is an internal team rotation over the years, as new teams are created, merged or split, with knowledge sharing when teams change, reducing the diversity of skills between author and reviewers and affecting the number of questions, doubts or different opinions. Surprisingly, the comment density by reviewer is higher when two teams are involved, followed by reviews involving one or three teams. The differences among teams are indeed significant ($H = 91.94, p < 0.05$), with post hoc tests showing that if more than three teams are involved, it actually makes no difference.

Conclusions of RQ-2: We found evidence that code review with more involved teams have lower effectiveness considering duration and participation, but higher effectiveness with respect to the overall comment density. Comment density by reviewer is slightly higher when two teams are involved when compared to reviews involving one or three teams.

5.4 RQ-3: locations

People involved in the code review are not only associated with a single team, but also with a single working site. We now investigate the influence of the number of involved locations on review outcomes. Results associated with this influence factor are shown in Table 6 and Fig. 4.

Table 6 Outcomes by number of locations

Locations	#Rev	DUR		PART		CD_G		CD_R	
		M	SD	M	SD	M	SD	M	SD
1	7219	4.1	10.8	80.8	24.1	45.5	98.4	22.5	46.8
2	1076	8.0	17.4	75.2	21.6	68.7	140.2	25.1	47.7
3	34	22.2	31.7	75.9	15.3	132.8	206.6	35.4	52.1
Total	8329	4.7	12.1	80.1	23.8	48.8	105.8	22.9	46.9

As can be seen, the duration of code review is considerably higher if more locations are involved. With a further analysis of our data, we observed that with two involved locations, reviews that started in the second half of the sprint sometimes were not finished on time, causing a performance penalty to the author's team—as said, code review is mandatory. This can be explained by the natural isolation of people working in different places, which requires daily effort to synchronize priorities and state the importance of every patch under review. Within the same team and location, this communication happens on a daily basis in the Scrum daily meetings or other activities that promote interaction. There is a statistically significant difference among the groups ($H = 158.0, p < 0.05$), in fact, among all groups, as shown in the post hoc analysis.

There is also a positive influence on comment density ($H = 134.05, p < 0.05$) and comment density by reviewer ($H = 56.12, p < 0.05$). However, there is a negative impact on participation ($H = 86.69, p < 0.05$). Post hoc tests show that for these outcomes the differences actually exist only between code reviews with one and two locations, probably because there are few occurrences involving three locations. One possible interpretation of these results, in addition to the geographical distance barrier, is that code reviews with more involved locations have more diversity of technical skills, which is plausible because teams are organized based on groups of related features and technologies. Moreover, there are few rotations of team members among different locations, creating some form of local technical specialization on each location. This diversity promotes feedback, questions, and comments, but requires more time to complete the review process. Consequently, reviewers from other locations should be invited if there is a good technical reason to do so. Otherwise, the higher duration is not compensated by a higher level of contributions.

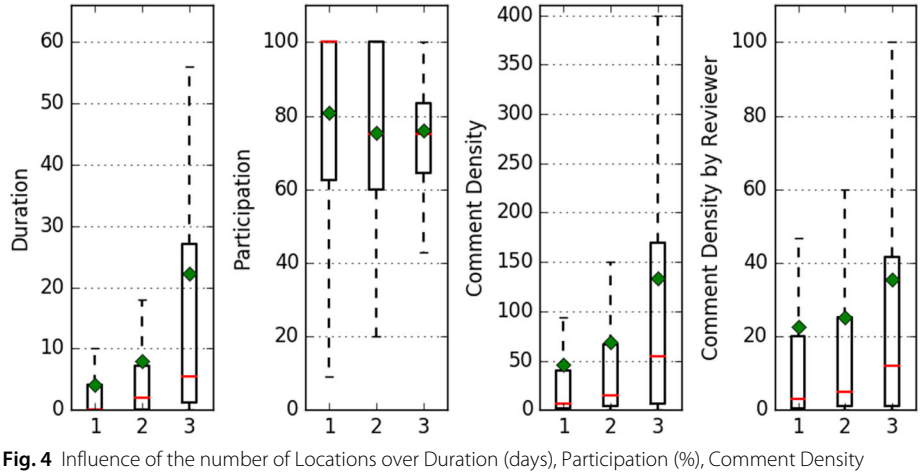

Fig. 4 Influence of the number of Locations over Duration (days), Participation (%), Comment Density (comments per 100 LOC) and Comment Density by Reviewer (comments per 100 LOC per active reviewers)

We also observed that the results with respect to comment density by reviewer have large differences when compared to those discussed in the previous sections. Results show that: (i) the average review duration in the same location is 32% greater than in the same team; (ii) the average duration with two locations is 38% greater than with two teams; and (iii) the average density of review comments with two locations is 24% higher than with two teams.

Conclusions of RQ-3: We found evidence that code reviews with more involved locations have lower effectiveness with respect to duration and participation, but higher effectiveness considering contributions. The overall comment density and comment density by reviewer are considerably higher with more involved locations. The participation is slightly lower with multiple involved locations.

5.5 RQ-4: active reviewers

As explained in Section 3, in the code review process, the only mandatory reviewer is the maintainer of the module. Other reviewers are invited, but their contribution is optional. Moreover, anyone can invite reviewers. Those that actually contribute are the *active reviewers*. The number of active reviewers might influence how other reviewers engage in the discussion, and this is what we investigate in RQ-4. Obtained results regarding the relationship between active reviewers and review outcomes are shown in Table 7 and Fig. 5—the group with 10–12 active reviewers are omitted in this figure because they have only one review occurrence each.

Considering duration, we intuitively expect higher values because more reviewers provide more feedback and comments and need more time to reach a consensus. This is in fact confirmed by a statistical test ($H = 1652.94, p < 0.05$), with post hoc tests showing that in most cases reviews with less than ten reviewers take more time to complete than with more active reviewers.

We highlight that there are reviews, more specifically 1313, involving one active reviewer. This occurs when the author is the module's maintainer, and thus is the only mandatory reviewer. Consequently, the duration is low in these cases, lasting 1.3 days on average. Exceptional cases occur when the code being reviewed is related to hardware

Table 7 Outcomes by number of active reviewers

Active	#Rev	DUR		PART		CD_G		CD_R	
Reviewers		M	SD	M	SD	M	SD	M	SD
1	2431	1.3	4.5	84.3	27.4	26.6	54.4	26.6	54.4
2	2502	3.2	7.6	78.6	24.8	54.5	100.5	27.2	50.2
3	1940	6.0	11.9	78.7	21.2	57.1	116.3	19.0	38.8
4	840	7.9	12.6	77.6	18.5	68.8	155.8	17.2	39.0
5	372	14.0	26.4	76.5	15.9	61.5	151.1	12.3	30.2
6	149	18.7	33.4	78.9	14.5	59.4	116.9	9.9	19.5
7	60	18.9	29.3	78.7	13.7	44.0	73.6	6.3	10.5
8	21	17.2	19.5	77.9	12.7	31.4	48.7	3.9	6.1
9	11	33.2	30.1	81.8	15.3	192.1	321.8	21.3	35.8
10	1	35.0	0.0	37.0	0.0	6.2	0.0	0.6	0.0
11	1	15.0	0.0	84.6	0.0	262.5	0.0	23.9	0.0
12	1	20.0	0.0	85.7	0.0	5.0	0.0	0.4	0.0
Total	8329	4.7	12.1	80.1	23.8	48.8	105.8	22.9	46.9

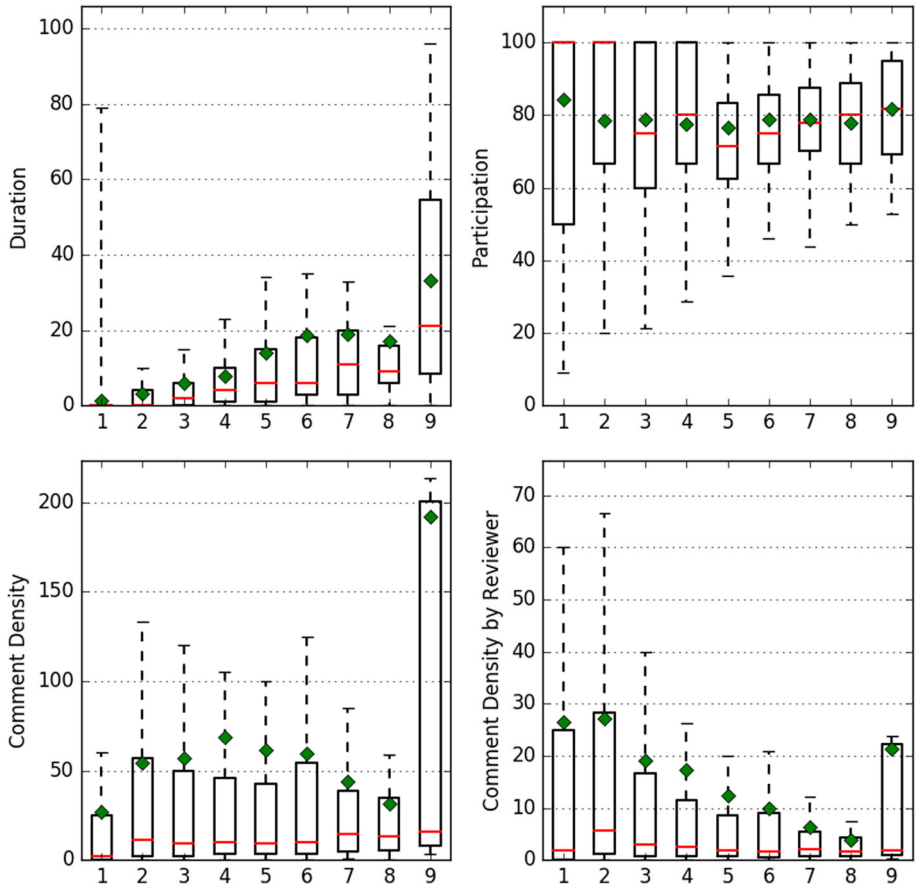

Fig. 5 Influence of the number of Active Reviewers over Duration (days), Participation (%), Comment Density (comments per 100 LOC) and Comment Density by Reviewer (comments per 100 LOC per active reviewers)

platforms and their infrastructure modules, where typically one or two developers work on for several months.

Reviewer participation is almost the same with more active reviewers. Although there are statistically significant differences among groups ($H = 268.49, p < 0.05$), the post hoc tests show that this is due to a few groups that have nearly no significant differences, indicating that the more invitees, the more active reviewers.

Considering the overall comment density, there is a statistically significant difference ($H = 660.89, p < 0.05$) when reviewers contribute. However, the post hoc tests show that the presence of more than two active reviewers does not significantly improve the comment density. Moreover, the comment density by reviewer is actually lower with three or more active reviewers ($H = 275.55, p < 0.05$). This suggests that a number of two active reviewers seems to be the optimal case considering a trade-off between duration and contributions from reviewers.

Conclusions of RQ-4: We found evidence that code review with more active reviewers has lower effectiveness considering the duration. The participation is slightly lower with more active reviewers. Moreover, having more than two active reviewers does not improve the overall comment density and negatively affects the comment density by reviewer.

6 Results and analysis: survey

Given that we presented the results of the part of our study that is based on objective data, now we focus on subjective data collected from our survey. This allows us to understand developers' perceptions of the relationship between influence factors and outcomes as well as contrast results obtained from objective and subjective data. As above, each influence factor is analyzed individually, with the aim to answer our research questions.

Before we focus on our first influence factor, namely Patch Size (LOC), we present an overview of the answers provided by participants in Fig. 6. Each box shows the proportion of the different answers provided by participants with respect to each pair of influence factor and outcome. Boxes that are concentrated on the left-hand side of the chart indicate that participants consider that there is an inversely proportional relationship between the influence factor and outcome. Boxes concentrated on the right-hand side, in contrast, indicate a directly proportional relationship among them.

In next sessions, we present for each research question a table with statistics about participants' responses, presenting minimum, maximum, median, average and standard deviation values. As Likert scales provide ordinal data, researchers have divergent opinions (Jamieson et al. 2004; Norman 2010) about the use of average and standard deviation values to summarize these data, but we present them as a complementary

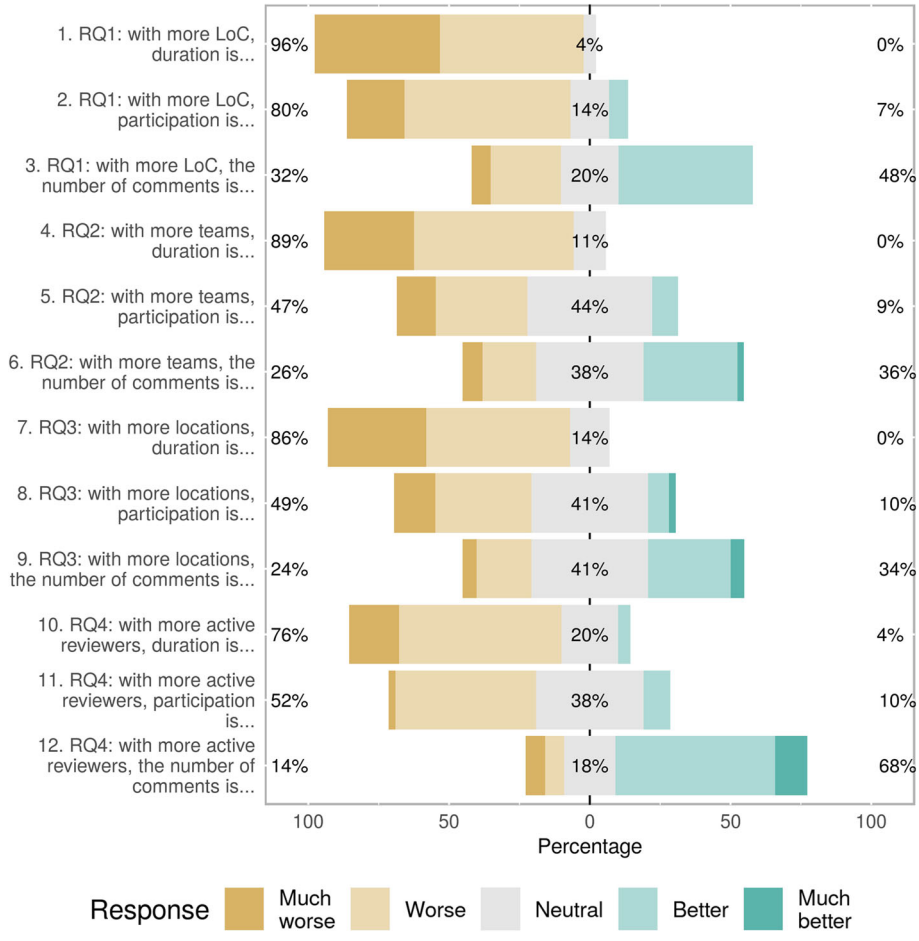

Fig. 6 Evaluation by participants of the relationship between influence factors and outcomes

description of data. Moreover, median and average values are similar considering our data, supporting the use of average to describe our data.

6.1 RQ-1: patch size (LOC)

The first three rows of Fig. 6 show that 96% of the participants believe that duration is negatively affected by patches with a higher number of LOC. Similar results are associated with participation, as 80% of participants stated that it becomes (much) worse. However, only 29% of the participants believe that the number of comments decreases for patches with more LOC, while 49% have a different opinion, indicating that more comments are provided.

Table 8 provides complementary details of the responses, including the number of participants that were unable to evaluate the influence in certain outcomes. For both duration and participation, average and median values are close to worse (2), with a high concentration of responses around this value. The number of comments, in contrast, has average and median values closer to no influence (3), but the standard deviation is higher, with a wider range of values. This suggests that some of the code review outcomes are more affected than others. To analyze differences among the effects on different outcomes, we conducted a Friedman's test, a non-parametric test as the distribution of values is not normal. The test revealed a statistically significant difference among groups ($\tilde{\chi}^2 = 54.47, p < 0.05$). We then used Nemenyi's test for post hoc tests, which indicated a significant difference among all groups. Consequently, we conclude that duration is the most affected outcome in the opinion of the participants, while the number of comments is the least affected.

6.2 RQ-2: Teams

Similarly to the results above, duration is negatively affected by an increased number of teams, as shown in Fig. 6. However, with respect to participation and number of comments, participants have divergent opinions. For participation, their opinions are divided between negative influence and no influence, while for number of comments there is a similar number of answers for all possible relationships (negative, positive or no influence).

These divergences among opinions can also be seen in Table 9, which shows the number of participants who answered each alternative. While duration have both average and median values close to worse (2), participation and number of comments have these values closer to no influence (3).

To verify if there is a code review outcome that is the most affected, we conducted similar statistical tests as above. A Friedman's test revealed a significant difference among groups ($\tilde{\chi}^2 = 47.73, p < 0.05$), with the post hoc tests showing differences among all

Table 8 Answers of participants evaluating the influence of patch size (LOC) on review outcomes

Outcome	Much worse (1.0)	Worse (2.0)	No Infl. (3.0)	Better (4.0)	Much better (5.0)	Unable to Resp.	Min	Max	Med	M	SD
Duration	20	23	2	0	0	0	1.0	3.0	2.0	1.6	0.6
Participation	9	26	6	3	0	1	1.0	4.0	2.0	2.1	0.8
Comments	3	11	9	21	0	1	1.0	4.0	3.0	3.1	1.0

Table 9 Answers of participants evaluating the influence of teams on review outcomes

Outcome	Much worse (1.0)	Worse (2.0)	No Infl. (3.0)	Better (4.0)	Much better (5.0)	Unable to Resp.	Min	Max	Med	M	SD
Duration	14	25	5	0	0	1	1.0	3.0	2.0	1.8	0.6
Participation	6	14	19	4	0	2	1.0	4.0	3.0	2.5	0.9
Comments	3	8	16	14	1	3	1.0	5.0	3.0	3.0	1.0

outcomes. This suggests that the duration is, once again, the most affected outcome in the opinion of the participants.

6.3 RQ-3: Locations

By comparing rows 4–6 to rows 7–9 in Fig. 6, it is possible to observe that results regarding the influence of teams and locations present the same pattern of answers. Table 10, in fact, shows that the number of answers for each alternative as well as average and median values are similar to those presented in Table 9. Consequently, in summary, participants also believe that a higher number of involved locations has a negative influence on duration, but have divergent opinions regarding the other two influence factors. Statistical tests also indicated the same results. A Friedman's test ($\tilde{\chi}^2 = 43.29, p < 0.05$) revealed a significant difference among groups, with the post hoc tests also pointing out duration as the most affected outcome.

6.4 RQ-4: active reviewers

Finally, the results associated with active reviews suggest that the higher the number of active reviewers, the worse the duration and participation, being the average associated with the latter closer to no influence. This can be seen in Fig. 6. Differently from the previous analyzed influence factors, participants believe that the number active reviewers have a positive influence on the number of comments (Table 11).

By conducting the same statistical tests used to identify significant differences among outcomes, we concluded that there is a significant difference among groups ($\tilde{\chi}^2 = 37.21, p < 0.05$). Moreover, as it is the case of patch size, this is due to differences across all groups. Consequently, duration is the most negatively affected outcome, while the number of comments is positively affected.

7 Discussion

In this section, we present insights and lessons learned from both studies and compare them. Finally, we discuss the threats to validity.

Table 10 Answers of participants evaluating the influence of locations on review outcomes

Outcome	Much worse (1.0)	Worse (2.0)	No Infl. (3.0)	Better (4.0)	Much better (5.0)	Unable to Resp.	Min	Max	Med	M	SD
Duration	15	22	6	0	0	2	1.0	3.0	2.0	1.8	0.7
Participation	6	14	17	3	1	4	1.0	5.0	3.0	2.5	0.9
Comments	2	8	17	12	2	4	1.0	5.0	3.0	3.1	0.9

Table 11 Answers of participants evaluating the influence of active reviews on review outcomes

Outcome	Much worse (1.0)	Worse (2.0)	No Infl. (3.0)	Better (4.0)	Much better (5.0)	Unable to Resp.	Min	Max	Med	M	SD
Duration	8	26	9	0	2	0	1.0	4.0	2.0	2.1	0.7
Participation	1	21	16	4	0	3	1.0	4.0	2.0	2.5	0.7
Comments	3	3	8	25	5	1	1.0	5.0	4.0	3.6	1.0

7.1 Lessons learned and comparison of results

Our study involves two parts (repository mining and survey) that aimed to answer the same research questions. The goal of performing the two analyses is to have two sources of evidence (one objetive and other subjective) to answer our research questions. Moreover, the contrast between the opinion of developers and our conclusions from the repository mining might reveal that developers may be unaware of the consequences of inviting reviewers from another team or location.

Before we start our discussion, we summarize in Table 12 the derived conclusions regarding how our investigated factors influence review outcomes, considering our investigated scenario in both studies. We highlight conflicting results.

7.1.1 Patch size

As can be seen, the amount of LOC to be reviewed affects all considered outcomes based on both sources of data. Considering the analysis of duration based on our objective data, we found that it is not linearly proportional to the patch size, as depicted in Fig. 7, which uses a scatter plot and a heat map to illustrate the concentration of data. This suggests that the rate of 200 LOC/hour proposed by Kemerer and Paulk (2009) is not followed, potentially making code review less effective and more error prone. A lower review coverage is another possible explanation to justify these results, which may lead to more post release defects (Shimagaki et al. 2016; McIntosh et al. 2014).

The results of the two parts of our study are consistent, because they both indicated negative effects of LOC on patch size on duration and participation. However, the perceived effect on the number of comments is close to neutral. This diverges from our conclusions from the repository mining, which indicated that fewer comments are provided for larger patches, so participants underestimate the possible harmful effects of larger patches in the discussion. Interestingly, one of the participants pointed out that after a certain number of LOC, the duration would likely decrease, causing reduced review quality. This was actually observed in our objective data for patches with more than 3 KLOC.

This suggests that large patches should be avoided and that further investigation is necessary to determine which kinds of tasks are likely to produce larger patches, and possibly

Table 12 Comparison of results based on objective (Obj.) and subjective (Subj.) data

Influence	Duration		Participation		Comments		
Factor	Obj.	Subj.	Obj.	Subj.	Obj. (CD_G)	Subj.	Obj. (CD_R)
Patch Size (LOC)	↑	↑	↓	↓	↓	↕	↓
Teams	↑	↑	↓	↓	↕	↕	↓
Locations	↑	↑	↓	↓	↑	↕	↑
Active reviewers	↑	↑	↓	↓	↕	↑	↕

Differences are highlighted. CD_R has no corresponding subjective measurement

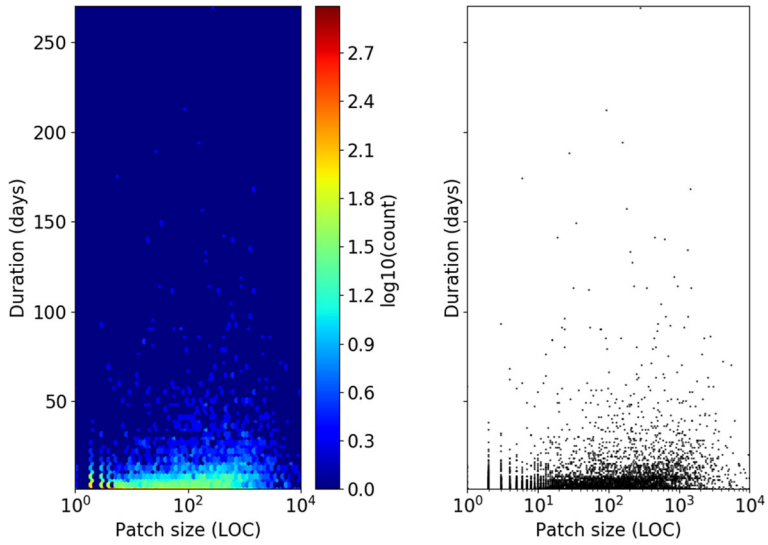

Fig. 7 Distribution of data considering Duration (days) and Patch Size (LOC)

use other forms of code review, like pair programming, to reduce the severe penalty over the duration.

7.1.2 Teams and locations

Regarding the effects of the number of involved teams, both objective and subjective data indicate an agreement of the negative effects on duration. This agreement also holds for participation. Nevertheless, the average perception (i.e. no influence) of the effect over the number of comments is different from that obtained with the repository mining, which suggested that more comments are provided when more teams are involved.

The effects of the number of locations, considering both parts of the study, on duration and participation are also negative. However, concerning the number of comments, a higher number of comments is expected when more locations are involved, according to the study based on repository mining, as opposed to no influence reported by participants of the survey. When comparing the effects of teams and locations, the study based on repository mining suggested that discussions are more fruitful with multiple locations involved, while the results of our survey suggest that there is no influence of teams and location on the number of comments.

The consistent effects of teams and locations over duration and participation might be associated with the lack of awareness of other teams' priorities, dependencies and schedules. The definition of collocation adopted by Olson and Olson (2000) is that teams are collocated if its members can reach each other with a short walk of 30 meters, suggesting that even people working in the same building or on the same floor are subject to the same issues. This shows that developers should carefully consider who from other teams and locations they invite as reviewers, and that they should put extra effort to ensure that external reviewers are available to do their job in reasonable time; collaboration readiness was in fact reported as a major factor on DSD (Olson et al. 2008; Olson and Olson 2013; Bjørn et al. 2014).

7.1.3 Active reviewers

The results obtained based on the mined project data give evidence that code review with more involved (active) reviewers are less effective, with very significant drawbacks on the average duration and density of review comments per reviewer. Results obtained based on our survey are consistent with this finding considering duration and participation. However, the results of our survey showed that participants often overestimate the positive effects of active reviewers to the total number of comments when more active reviewers are present, as 68% participants reported that the number of comments is (much) higher when more reviewers are active, while the study based on repository mining suggested that having more than two reviewers does not improve discussion. This suggests, again, the importance of choosing adequate reviewers and calls for sophisticated code reviewer recommenders.

7.2 Contributions

On the contributions of this paper, considering the related work on modern code review that we presented in Table 1, this work analyzed the influence of different influence factors, such as the number of teams, locations and active reviewers. The same holds for the code review outcomes, as these studies did not analyze the participation and comment density. Moreover, our mixed-method study evidentiated the dissonance between participants' perception and metrics extracted from code review databases in some situations. By collecting the opinion of participants using a form that is symmetric with the research questions, our study presented a significant degree novelty when compared to related work, as only Bosu et al. (2015) used insights obtained from interviews with developers create a definition for the usefulness of code review comments.

Code review practitioners can benefit from the discussion section, which was based on the analysis of results from both parts of this study and should foster critical thinking on simple, daily decisions inside the teams. Although some of the suggestions are relatively simple to adopt, the existence of the problem itself is not always evident. For instance, having more invited reviewers does not mean that the code will be reviewed faster or that more comments will be provided, so authors should carefully select reviewers. When reviewers from other teams or locations are involved, the likelihood of having extra delays is something that the teams should be aware of, and eventually adopt countermeasures instead of just assuming that review is in progress. Similarly, delivering smaller patches improves the code review process, so smaller tasks are preferable.

7.3 Threats to validity

Construct validity As mentioned in Section 3, our target project involves many programming languages, including Yang, which is a way to represent data. When counting and analyzing lines of code, code written in all these languages are treated equally. Reviewing the same amount of code in one language may require more time than in another. However, considering the developers' expertise, they do not state more difficulty in reviewing code in particular languages, and also the involved languages are not largely different with respect to verbosity. Furthermore, many medium and large software projects use many programming languages. Therefore, the amount of code in different languages is considered a random variable rather than a confounding variable of the study.

Considering our survey, we adopted a single variable, total number of comments, instead of the complementary metrics of the first part of the study, namely Comment Density per 100 LOC and Comment Density per 100 LOC by Reviewer. Although complex metrics help the analysis of influence of factors on comments, we assumed that a single variable would be more adequate for developers to evaluate in the survey. The adequacy of this choice was confirmed by our pilot study. However, this may cause the comparison between the results of the repository mining and the survey to be less accurate.

Internal validity We identified five internal threats. First, given that we analyzed an extensive period of our target project, its developers changed over time. However, as the number of developers and analyzed reviews is large, individual developers' behavior and expertise have a low impact on the obtained results. Moreover, this change in development teams is expected in any software project.

Second, in most of the cases, authors and reviewers communicate using Gerrit to provide feedback, even when they are on the same team or location. However, there is no explicit obligation in the target project to record in Gerrit feedback given by means of other forms of communication, such as telephone or informal meetings. Nevertheless, this is very unusual for this project—developers tend to use the available tools to ensure that relevant questions will not be forgotten by the authors. Isolated occurrences thus do not significantly affect the results.

Third, the participation outcome may have been affected due to the automatic addition of reviewers by Gerrit's reviewers-by-blame plugin. The plugin may add, as reviewers, developers that no longer work in the same team or even in the company. Consequently, their participation was not expected. As what matters is the relative comparison of participation for groups of each outcome, this likely has not affected the results. The probability of having reviewers that fall into this category is the same for the different reviews.

Fourth, our survey was conducted after the publication of the shorter version of this work (Witter dos Santos and Nunes 2017). If participants had access to this work before taking part of the survey, they may have been influenced by our previous results. Although we cannot completely confirm that no participant became aware of the work, no results were intentionally disclosed within the company in which the participants work. Moreover, the time gap between the earlier version of this work and the collection of survey data is only of three months.

Finally, our survey was conducted with participants from the same company on which we collected data for the objective study, but from potentially different projects. Moreover, participants of the survey did not necessarily participate of the project that had its data analyzed. However, all participants use the same code review process and the same tools, including the automated reviewers.

External validity Generalizing the results of empirical studies is always an issue, because the collected and analyzed data may not generally represent software projects. Although we focused on a single project, our results are based on a large amount of data of a large project. Therefore, we were able to identify trends and statistically significant results. However, we emphasize that our results are potentially generalizable only for distributed

development environments similar to that of our target project. Although geographically distributed, development locations occur in the same country and mostly involve developers of the same nationality. Therefore, further studies should investigate whether our results hold to *globally* distributed development environments, which may impose additional barriers to MCR, such as different time zones, communication languages, and culture.

8 Conclusion

Code review is an important static verification technique for improving software quality as well as promotes knowledge sharing within a software project. To identify the scenarios in which code review in fact succeeds, many studies investigated the relationship between different factors and the review outcomes. However, there is limited investigation of the situations in which modern code review is effective in the context of distributed software development when developers and reviewers are spread into geographically distant development locations.

In this paper, we presented the results of a mixed-method study, composed of two parts. In the first part, repository mining, we extracted a large amount of code review information from a software project whose aim is to develop an operating system for embedded systems. This project involves 201 developers, spread into 21 teams located in 4 different cities. We investigated how the patch size (in terms of lines of code), the number of teams, the number of locations and the number of active reviewers influence the duration, reviewer participation and comment density (general and by reviewer) of the review. We found evidence that the duration of the code review is highly affected by all investigated factors—the higher they are, the longer the review process. Similarly, the participation of reviewers is negatively affected in all cases, but mainly by the number of lines of code to be reviewed. The density of review comments is higher when a relatively small patch size is reviewed by other reviewers of teams or locations other than that of the author. The density of review comments per reviewer is positively affected by the number of involved locations and negatively affected by the other factors.

In the second part of the study, we conducted a survey to collect data about the perceived effects of the four investigated influence factors over code review outcomes (duration, participation and total number of comments). We obtained 50 responses from software developers with relevant professional experience in DSD projects with modern code review practices. We found evidence that higher values of the influence factors have similar effects on the analyzed code review outcomes. Duration and participation are negatively affected; the total number of comments is negatively affected by patch size, teams and locations, but is positively affected by the number of active reviewers.

Due to the large amount of data investigated in our study, we could not identify particular occurrences of code review that could help us to make other analyses and further explain our data. Even if this was possible, given that we used data from the past to have complete reviews, developers would potentially not remember specific cases. Our study had, however, gave us insights for future investigations. First, we aim to perform an observational study involving developers and managers that will allow us to verify if our conclusions based on the present study hold. Second, further analyses can be made using code review data. For example, the proportion of votes (vetoes, rejections, approvals and neutral feedback), the influence of the number of contributions as author or reviewer

(overall and in the same module or file) and other reviewers' characteristics are interesting issues to be investigated.

As the patch size demonstrated to be a prominent influence factor, we also plan to analyze other forms of complexity and effort during code review, assuming that reviewing ten lines added to a complex module requires more effort than reviewing ten lines added to a simple module. It is possible to analyze the influence of other indications of complexity, such as the total number of classes, files and LOC as well as the total cyclomatic complexity.

For modular systems, some influence factors arise from the relations among the modules and from the role of each module. For instance, the number of dependent modules could influence the participation in or the duration of the code review, and critical infrastructure modules might have different code review dynamics when compared to modules that implement user interfaces. Therefore, we plan to analyze the influence of architectural aspects of the modules in the code review.

Finally, we considered many metrics to indicate the effectiveness of the review and aim to investigate whether it is possible to derive a single metric that captures review effectiveness by combining different review outcomes.

Endnotes

[1] https://www.gerritcodereview.com/

[2] Available at http://www.inf.ufrgs.br/prosoft/resources/2017/sbes-mcr-dsd.

Abbreviations

DSD: Distributed software development; FLOSS: Free, libre and open source software; FPS: File path similarity; GQM: Goal-question-metric; LOC: Lines of code; M: Mean; Max: Maximum; MCR: Moderm code review; Med: Median; Min: Minimum; RQ: Research question; SD: Standard deviation

Funding

Ingrid Nunes would like to thank for research grants CNPq ref. 303232/2015-3.

Availability of data and materials

The original data used in repository mining phase of this study cannot be disclosed due to a confidentiality agreement; even anonymized or obfuscated data may be used to deduct more than we are allowed to disclose. Regarding the survey with software developers about code review, the answers of each individual participant are not available also due to the term of agreement signed by each participant that participated in the survey. The scripts used for repository mining can be provided upon request to the authors to allow further replications. However, this is conditioned to the approval from the company involved in the study to ensure that no sensible data is accidentally disclosed.

Authors' contributions

EW and IN worked jointly on the conceptualization of the work as well as analysis and interpretation of the collected data. The paper was written in an interactive way, receiving contributions of both authors. EW was the leader, writing first drafts and making initial analyses, and also being responsible for the implementation needed to collect the data, data collection, and execution of statistical tests. Both authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Received: 16 February 2018 Accepted: 7 October 2018
Published online: 26 October 2018

References

Bacchelli A, Bird C (2013) Expectations, Outcomes, and Challenges of Modern Code Review. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13. IEEE Press, Piscataway. pp 712–721. http://dl.acm.org/citation.cfm?id=2486788.2486882

Balachandran V (2013) Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13. IEEE Press, Piscataway. pp 931–940. http://dl.acm.org/citation.cfm?id=2486788.2486915

Basili VR, Selby RW, Hutchens DH (1986) Experimentation in software engineering. IEEE Trans Softw Eng 12(7):733–743
Baysal O, Kononenko O, Holmes R, Godfrey MW (2016) Investigating technical and non-technical factors influencing modern code review. Empir Softw Eng 21(3):932–959
Beller M, Bacchelli A, Zaidman A, Juergens E (2014) Modern Code Reviews in Open-source Projects: Which Problems Do They Fix?. In: Proceedings of the 11th Working Conference on Mining Software Repositories. MSR 2014. ACM, New York. pp 202–211. http://doi.acm.org/10.1145/2597073.2597082
Bisant DB, Lyle JR (1989) A two-person inspection method to improve programming productivity. IEEE Trans Softw Eng 15(10):1294
Bjørn P, Esbensen M, Jensen RE, Matthiesen S (2014) Does distance still matter? revisiting the cscw fundamentals on distributed collaboration. ACM Trans Comput-Hum Interact (TOCHI) 21(5):27
Bosu A, Greiler M, Bird C (2015) Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In: Proceedings of the 12th Working Conference on Mining Software Repositories. MSR '15. IEEE Press, Piscataway. pp 146–156. http://dl.acm.org/citation.cfm?id=2820518.2820538
Czerwonka J, Greiler M, Tilford J (2015) Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2. ICSE '15. IEEE Press, Piscataway. pp 27–28. http://dl.acm.org/citation.cfm?id=2819009.2819015
Fagan ME (1976) Design and code inspections to reduce errors in program development. IBM Syst J 15(3):182–211. http://dx.doi.org/10.1147/sj.153.0182
Fagan ME (1986) Advances in software inspections. IEEE Trans Softw Eng 12(1):744–751
Ferreira AL, Machado RJ, Silva JG, Batista RF, Costa L, Paulk MC (2010) An Approach to Improving Software Inspections Performance. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance. ICSM '10. IEEE Computer Society, Washington. pp 1–8. http://dx.doi.org/10.1109/ICSM.2010.5609700
Google (2017a) Gerrit Code Review. https://www.gerritcodereview.com. Accessed 16 Oct 2018
Google (2017b) Gerrit Code Review v2.11.2, Queries. https://gerrit-documentation.storage.googleapis.com/Documentation/2.12.2/cmd-query.html. Accessed 16 Oct 2018
Google (2017c) Gerrit Plugin: Reviewers by Blame. https://gerrit-review.googlesource.com/Documentation/config-plugins.html#reviewers-by-blame. Accessed 16 Oct 2018
Hundhausen CD, Agrawal A, Agarwal P (2013) Talking about code: Integrating pedagogical code reviews into early computing courses. Trans Comput Educ 13(3):14:1–14:28
Internet Engineering Task Force (IETF) (2017) RFC 6020: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). http://tools.ietf.org/html/rfc6020
Jamieson S, et al (2004) Likert scales: how to (ab) use them. Med Educ 38(12):1217–1218
Kemerer CF, Paulk MC (2009) The impact of design and code reviews on software quality: An empirical study based on psp data. IEEE Trans Softw Eng 35(4):534–550
Kollanus S, Koskinen J (2009) Survey of software inspection research. Open Softw Eng J 3(1):15–34
Martin J, Tsai WT (1990) N-fold inspection: A requirements analysis technique. Commun ACM 33(2):225–232
McIntosh S, Kamei Y, Adams B, Hassan AE (2014) The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. In: Proceedings of the 11th Working Conference on Mining Software Repositories. MSR 2014. ACM, New York. pp 192–201. http://doi.acm.org/10.1145/2597073.2597076
Meyer B (2008) Design and code reviews in the age of the internet. Commun ACM 51(9):66–71
Norman G (2010) Likert scales, levels of measurement and the "laws" of statistics. Adv Health Sci Educ 15(5):625–632
Olson GM, Olson JS (2000) Distance matters. Human-computer Interact 15(2):139–178
Olson JS, Olson GM (2013) Working together apart: Collaboration over the internet. Synth Lect Human-Centered Inform 6(5):1–151
Olson JS, Hofer E, Bos N, Zimmerman A, Olson GM, Cooney D, Faniel I (2008) A theory of remote scientific collaboration. In: Olson GM, Zimmerman A, Bos N (eds). Scientific collaboration on the Internet. MIT Press, Cambridge. pp 73–99
Parnas DL, Weiss DM (1985) Active Design Reviews: Principles and Practices. In: Proceedings of the 8th International Conference on Software Engineering. ICSE '85. IEEE Computer Society Press, Los Alamitos. pp 132–136. http://dl.acm.org/citation.cfm?id=319568.319599
Perpich JM, Perry DE, Porter AA, Votta LG, Wade MW (1997) Anywhere, anytime code inspections: Using the web to remove inspection bottlenecks in large-scale software development. In: Proceedings of the 19th International Conference on Software Engineering. ICSE '97. ACM, New York. pp 14–21. http://doi.acm.org/10.1145/253228.253234
Rahman MM, Roy CK, Redl J, Collins JA (2016) CORRECT: Code Reviewer Recommendation at GitHub for Vendasta Technologies. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE 2016. ACM, New York. pp 792–797. http://doi.acm.org/10.1145/2970276.2970283
Sengupta B, Chandra S, Sinha V (2006) A Research Agenda for Distributed Software Development. In: Proceedings of the 28th International Conference on Software Engineering. ICSE '06. ACM, New York. pp 731–740. http://doi.acm.org/10.1145/1134285.1134402
Shimagaki J, Kamei Y, McIntosh S, Hassan AE, Ubayashi N (2016) A Study of the Quality-impacting Practices of Modern Code Review at Sony Mobile. In: Proceedings of the 38th International Conference on Software Engineering Companion. ICSE '16. ACM, New York. pp 212–221. http://doi.acm.org/10.1145/2889160.2889243
Stein M, Riedl J, Harner SJ, Mashayekhi V (1997) A case study of distributed, asynchronous software inspection. In: Proceedings of the 19th International Conference on Software Engineering. ICSE '97. ACM, New York. pp 107–117. http://doi.acm.org/10.1145/253228.253250
Thongtanunam P, Kula RG, Cruz AEC, Yoshida N, Iida H (2014) Improving Code Review Effectiveness Through Reviewer Recommendations. In: Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering. CHASE 2014. ACM, New York. pp 119–122. http://doi.acm.org/10.1145/2593702.2593705
Thongtanunam P, McIntosh S, Hassan AE, Iida H (2015a) Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System. In: Proceedings of the 12th Working Conference on Mining Software Repositories. MSR '15. IEEE Press, Piscataway. pp 168–179. http://dl.acm.org/citation.cfm?id=2820518.2820540

Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto Ki (2015b) Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In: SANER 2015. IEEE. pp 141–150

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2016a) Review participation in modern code review. Empir Softw Eng 22(2):768–817

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2016b) Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In: Proceedings of the 38th International Conference on Software Engineering. ICSE '16. ACM, New York. pp 1039–1050. http://doi.acm.org/10.1145/2884781.2884852

Viviani G, Murphy GC (2016) Removing Stagnation from Modern Code Review. In: Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity. SPLASH Companion 2016. ACM, New York. pp 43–44. http://doi.acm.org/10.1145/2984043.2989224

Witter dos Santos E, Nunes I (2017) Investigating the effectiveness of peer code review in distributed software development. In: Proceedings of the 31st Brazilian Symposium on Software Engineering. SBES'17. ACM, New York. pp 84–93. http://doi.acm.org/10.1145/3131151.3131161

Xia X, Lo D, Wang X, Yang X (2015) Who Should Review This Change?: Putting Text and File Location Analyses Together for More Accurate Recommendations. In: Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). ICSME '15. IEEE Computer Society, Washington. pp 261–270. http://dx.doi.org/10.1109/ICSM.2015.7332472

Yang X (2014) Social Network Analysis in Open Source Software Peer Review. In: Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE 2014. ACM, New York. pp 820–822. http://doi.acm.org/10.1145/2635868.2661682

Zanjani MB, Kagdi H, Bird C (2016) Automatically recommending peer reviewers in modern code review. IEEE Trans Softw Eng 42(6):530–543